Análisis del Uso de un Cluster de Raspberry Pi para Cómputo de Altas Prestaciones

Pablo Sebastián Rodríguez Eguren

Facultad de Informática

Universidad Nacional de La Plata

Director: Esp. Franco Chichizola

Director: Dr. Enzo Rucci

Tesina presentada para obtener el grado de

Licenciado en Sistemas

Mayo 2018

Agradecimientos

A mis padres Heber y Mabel, a mi hermano Mauro y a mi hermana Daiana, por el esfuerzo y acompañamiento que me brindaron a lo largo de la carrera;

a mi novia Vero por su cariño, comprensión y por incentivarme a seguir adelante;

a mis compañeros y directores, Franco y Enzo, por la paciencia, por dirigirme y hacer esta tesina posible;

al instituto III-LIDI por recibirme, rodearme de colegas de los cuales aprendo día a día, y por brindarme los medios y recursos para realizar esta tesina.

Índice general

Índice de figuras

Capítulo 1

Introducción

Se presenta la motivación de esta tesina, y se detallan: el objetivo, la metodología, los desarrollos propuestos y los resultados esperados. Finalmente, se describe la organización de la tesina.

1.1. Motivación

En la actualidad, los grandes sistemas de cómputo de altas prestaciones (HPC, por sus siglas en inglés) son dominados por procesadores de propósito general con dos conjuntos de instrucciones (x86 y Power) de tres empresas vendedoras (Intel, AMD e IBM) [1]. Estos procesadores han sido diseñados especialmente para ofrecer un alto rendimiento por núcleo, pero con costo económico (de cientos a miles de dólares) y demanda de potencia acordes (en el rango de 60-140W). En la búsqueda de construir sistemas más grandes que nos permita llegar a la escala de los Exaflops, resulta fundamental explorar otras estrategias que ofrezcan tasas de rendimiento costo-eficientes y que sean capaces de reducir el consumo energético de estos sistemas [2].

Durante décadas, el desarrollo de plataformas HPC estuvo focalizado casi únicamente en mejorar el rendimiento de las mismas. Esto provocó un crecimiento exponencial en los requerimientos de potencia de estos sistemas (no sólo para alimentarlos sino también para refrigerarlos), lo que a su vez repercutió en el costo económico de los mismos. Es por eso que, hoy en día, la reducción del consumo energético se ha vuelto unos de los principales desafíos para la comunidad HPC.

Entre las diferentes alternativas actualmente exploradas, se encuentra el uso de procesadores de bajo consumo como los ARM. Esta clase de procesadores ha sido especialmente diseñada para el mercado de dispositivos móviles y los sistemas em-

bebidos. Recientemente, los procesadores ARM han entrado al mercado de PCs y servidores y se espera que en el futuro incrementen significativamente su potencia de cómputo manteniendo el bajo consumo de potencia [3]. Para la comunidad HPC, este escenario presenta una oportunidad de emplear los procesadores ARM para construir sistemas de alto rendimiento.

En los últimos años, el uso de procesadores ARM se ha vuelto popular entre desarrolladores y entusiastas de la programación debido al surgimiento de computadoras de placa única (SBC, por sus siglas en inglés: Single Board Computer) como la Beagleboard [4], la Raspberry Pi [5] y la Pandaboard [6]. Esto ha llevado a la formación de varias comunidades de desarrollo que invierten tiempo y esfuerzo en portar una amplia gama de sistemas operativos, aplicaciones, librerías y herramientas a los sistemas ARM más populares [7].

La Raspberry Pi (RPi) es una SBC de bajo costo desarrollada en el Reino Unido por la Fundación Raspberry Pi, con el objetivo de estimular la enseñanza de las ciencias de la computación en las escuelas. Desde su surgimiento, la RPi ha sido adoptada masivamente para una amplia gama de aplicaciones debido a su alto cociente rendimiento/precio, su tamaño reducido (similar al de una tarjeta de crédito) y su versatilidad [8]. En particular, desde el punto de vista del procesamiento paralelo, las RPi tienen múltiples núcleos, pueden conectarse en red y dan soporte a sistemas operativos y herramientas de programación tradicionales de HPC. Aunque sus núcleos no son potentes, el bajo consumo de potencia de las RPi las vuelven una opción atractiva desde el punto de vista energético. Es por eso que resulta interesante analizar la viabilidad del uso de un clúster de RPi para HPC.

1.2. Objetivo y metodología

El objetivo de esta tesina consiste en armar un cluster de placas RPi y analizar la viabilidad de su uso para cómputo de altas prestaciones. Para ello se realizarán las siguientes actividades:

- se estudiarán los fundamentos del procesamiento paralelo,
- se analizará la factibilidad del armado de un cluster de RPi contemplando las distintas opciones de configuración (tanto a nivel de hardware como de software),
- se seleccionarán aplicaciones que posean alta demanda computacional y diferentes características en su ejecución,

- se evaluará el rendimiento y la eficiencia energética del cluster armado para las aplicaciones seleccionadas comparándolo con un cluster de multicores convencional,
- se comparará la propuesta llevada a cabo y sus resultados con otros existentes en la literatura,
- se realizará un análisis de la viabilidad del uso de cluster de RPi para HPC.

1.3. Desarrollos propuestos y resultados esperados

Esta tesis propone el despliegue de un cluster de placas RPi que sea capaz de ejecutar aplicaciones paralelas empleando software de base y herramientas tradicionales de HPC. Entre los resultados esperados podemos mencionar:

- estudio del armado de un cluster de placas RPi. Este estudio contemplará las opciones de configuración tanto desde el punto de vista del hardware como del software y resultará útil para cualquier individuo que desee armar un cluster de esta clase.
- evaluación de rendimiento y eficiencia energética sobre un cluster de RPi y su comparación con un cluster de multicores x86. Mediante la ejecución de aplicaciones paralelas con diferentes características, este estudio permitirá caracterizar las tasas de prestaciones y eficiencia energética que un cluster de RPi puede alcanzar.
- análisis de la viabilidad de un cluster de RPi para HPC. Este análisis resultará valioso para cualquier individuo que deba decidir sobre el despliegue de un cluster de RPi para su institución, empresa u organización.

1.4. Organización de la tesina

El resto del documento se organiza de la siguiente forma:

- en el Capítulo 2 se estudian los fundamentos del procesamiento paralelo. En particular, se describen los principios de HPC, las arquitecturas paralelas y sus características, los modelos y herramientas para programación paralela junto a las técnicas usuales de optimización. Por último, se mencionan diferentes métricas de rendimiento reconocidas.

- en el Capítulo 3 se estudian las placas RPi, las cuales fueron seleccionadas para el armado del cluster desplegado en esta tesina. En primer lugar, se enuncia la historia del surgimiento de las primeras placas RPi, cómo logra insertarse en el mercado y el porqué de su auge. A continuación se describen las características y la evolución de cada modelo de placa RPi. Además, se mencionan los distintos sistemas operativos y sus principales características. Finalmente, se realiza una comparación teórica con procesadores x86.

- en el Capítulo 4 se presenta el despliegue y puesta a punto del cluster RPi. Primeramente, se menciona el hardware utilizado y la descripción en detalle del mismo (placas, switch, fuente de alimentación, etc). Después, se describe paso a paso la configuración común para los nodos y luego para cada uno de los nodos en concreto (*master* y *worker*). Por último, se describen las pruebas de integración y prueba del cluster para verificar el correcto funcionamiento del mismo.

- en el Capítulo 5 se analiza el rendimiento del cluster desplegado para esta tesina. Inicialmente, se describen las aplicaciones seleccionadas para la evaluación de prestaciones, detallando sus características relevantes. A continuación, se plasman y estudian los resultados obtenidos a fin de analizar el comportamiento del cluster RPi. Además, se realiza una comparación del cluster desplegado con un cluster de multicores x86, tanto desde el punto de vista del rendimiento computacional como de la eficiencia energética. Finalmente, se analizan trabajos relacionados disponibles en la literatura.

- en el Capítulo 6 se presentan las conclusiones de esta tesina y se describen algunas de las posibles líneas de trabajo futuro.

Capítulo 2

Procesamiento paralelo

2.1. ¿Qué es el procesamiento paralelo y para qué se utiliza?

Si nos transportamos un par de décadas atrás, una computadora tradicional contaba con una unidad de procesamiento individual para ejecutar las instrucciones especificadas en un programa. Los problemas eran divididos en series discretas de instrucciones, donde estas eran ejecutadas de a una por vez, unas tras otras. Una manera de incrementar el poder de cómputo es usar más de una unidad de procesamiento dentro de una única computadora o bien utilizar varias computadoras, trabajando todas juntos sobre el mismo problema. El problema es dividido en partes discretas, donde cada una de ellas es resuelta por una unidad de procesamiento individual de manera paralela. Se puede definir al procesamiento paralelo como el uso simultáneo de múltiples recursos computacionales para resolver un problema[9].

La necesidad de la computación paralela se origina por las limitaciones de los computadores secuenciales: integrando varios procesadores para llevar a cabo la computación es posible resolver problemas que requieren de más memoria o de mayor velocidad de cómputo. El objetivo principal de la computación paralela es reducir el tiempo de resolución de problemas computacionales, o bien para resolver problemas más grandes que no podrían ser resueltos por un computador convencional. Para llevar a cabo esto, es necesario emplear sistemas de cómputo de altas prestaciones y algoritmos paralelos que utilicen estos sistemas eficientemente.

Los problemas habituales en los cuales se aplica la programación paralela son: problemas con alta demanda de cómputo, problemas que requieren procesar una gran cantidad de datos, o problemas de tiempo real, en los que se necesita la respuesta en

un tiempo máximo. De esta forma, la comunidad científica usa la computación paralela para resolver problemas que sin el paralelismo serían intratables, o con tiempos de respuesta inaceptables. Algunos campos que se favorecen de la programación paralela son: predicciones y estudios meteorológicos, estudio del genoma humano, modelado de la biosfera, predicciones sísmicas, simulación de moléculas, modelización y simulación financiera, computación gráfica, realidad virtual, motores de búsqueda web, exploración de hidrocarburos, diseño de fármacos, entre otros.

Aunque tradicionalmente el objetivo primario de HPC fue reducir el tiempo de ejecución de las aplicaciones, en las últimas décadas la eficiencia energética ha cobrado un valor semejante. Esto se debe al elevado consumo energético y generación de calor de las arquitecturas paralelas que afectan al funcionamiento y repercuten en el costo, debido a la adquisición de equipos para refrigeración y al consumo energético que generan los mismos.

2.2. Dificultades

Existen diversas dificultades que se pueden encontrar a la hora de escribir un programa paralelo. Por ejemplo, no siempre es posible paralelizar un programa; la paralelización requiere de tareas que no están presentes en la programación secuencial (descomposición del problema, comunicación y sincronización, mapeo, entre otras); mayor propensión a cometer errores por aumento de la complejidad; mayor dificultad a la hora de probar o depurar un programa; y por último, la fuerte dependencia entre el programa paralelo y la arquitectura de soporte para obtener alto rendimiento.

2.3. Plataformas paralelas

Una plataforma paralela consiste de dos o más unidades de procesamiento vinculadas a partir de algún tipo de red de interconexión. Es posible clasificarlas de dos formas, según su organización lógica o según su organización física. Se entiende como organización lógica, a la manera en que el programador visualiza la plataforma paralela. Por otro lado, la organización física se refiere al hardware real de la plataforma. A continuación se describen los diferentes tipos de plataformas paralelas.

2.3.1. Plataformas de memoria compartida

Un multiprocesador de memoria compartida consiste en dos o más unidades de procesamiento conectadas a múltiples módulos de memoria. La forma de conexión de estos elementos es a través de algún tipo de red de interconexión como puede ser un switch crossbar o un bus. En estos sistemas, existe un único espacio de direcciones, por lo que todas las unidades de procesamiento pueden acceder a la misma dirección de memoria, y modificar los datos almacenados en este espacio compartido. Si todas las unidades de procesamiento tienen el mismo tiempo para acceder a cualquier dirección de memoria, entonces se dice que el multiprocesador tiene acceso uniforme a memoria (UMA). Por el contrario, si el tiempo de acceso a algunas direcciones de memoria es mayor que a otras, entonces se dice que el multiprocesador es de acceso no uniforme a memoria (NUMA) [10, 11].

Idealmente, se desea que el sistema sea UMA. No obstante, los grandes sistemas de memoria compartida suelen ser de tipo NUMA dada la dificultad de implementar hardware que provea un acceso rápido a toda la memoria compartida. Por lo que cuentan con alguna estructura de memoria jerárquica o distribuida [9].

Tanto los sistemas UMA como los NUMA, cuentan con una memoria caché de alta velocidad para mantener los contenidos recientemente referenciados de las direcciones de memoria principal. A pesar de obtener mejoras con la presencia de cachés en las unidades de procesamiento, este tipo de memoria también acarrea la problemática de tener múltiples copias de una única palabra de memoria siendo manipulada por más de una unidad de procesamiento al mismo tiempo. Un ejemplo de este tipo de arquitecturas son los multicores.

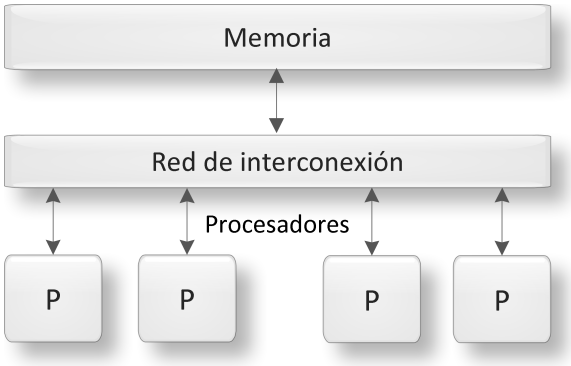


Figura 2.1: Plataforma de memoria compartida.

2.3.2. Plataformas de memoria distribuida

Una plataforma de memoria distribuida consiste de varios nodos de procesamiento independientes con módulos de memoria locales, esto significa que cada uno cuenta con su propio espacio de direcciones. Además, estos nodos están conectados por una red de interconexión. Cada nodo puede ser una computadora individual o un multiprocesador de memoria compartida. Al contar con su propio espacio de memoria, el mismo no es accesible por el resto y los nodos deben comunicarse entre si enviándose mensajes. Éste intercambio de mensajes es utilizado para transferir datos, trabajo y sincronizar acciones entre nodos [11].

A la hora de escalar físicamente alguna de estas plataformas, resulta más fácil hacerlo en los sistemas de memoria distribuida que en los sistemas de memoria compartida [9].

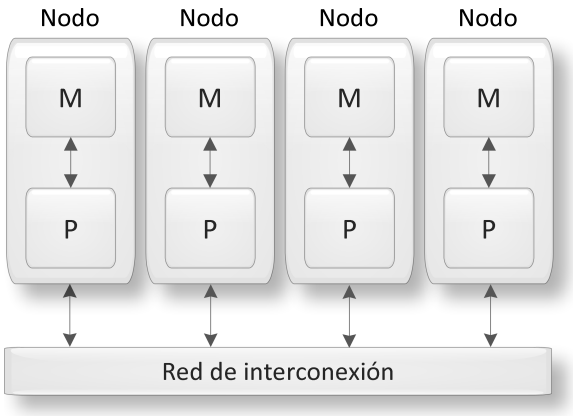


Figura 2.2: Plataforma de memoria distribuida.

2.3.3. Plataformas híbridas

Con la incorporación de los procesadores multicore a las arquitecturas de clusters tradicionales surgió una nueva plataforma híbrida conocida como cluster de multicores [12]. Con la aparición de esta plataforma, surgen nuevos tipos de comunicación debido a la heterogeneidad de la arquitectura [13], que se clasifican en: inter-nodo e intra-nodo. El primero se da entre los núcleos de distintos nodos y se lleva a cabo mediante envío de mensajes a través de la red de interconexión. El segundo se da entre los distintos núcleos que tiene un nodo y se lleva a cabo a través la jerarquía de memoria que estos núcleos comparten.

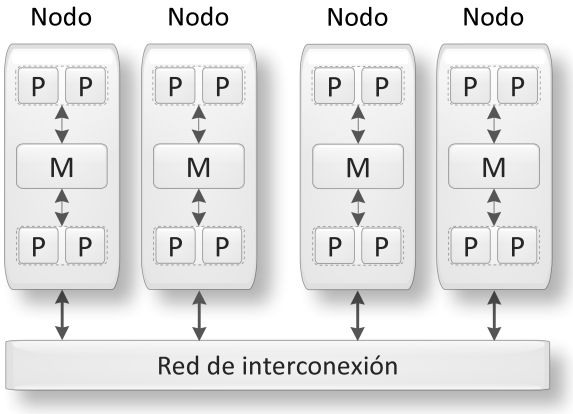


Figura 2.3: Plataforma de memoria híbrida.

2.3.4. Taxonomía de Flynn

Las computadoras paralelas se han utilizado durante muchos años, y se han propuesto y utilizado muchas alternativas arquitectónicas diferentes. En general, una computadora paralela se puede caracterizar como una colección de elementos de procesamiento que pueden comunicarse y cooperar para resolver rápidamente grandes problemas [14]. Esta definición es intencionalmente bastante vaga para capturar una gran variedad de plataformas paralelas. La definición no aborda muchos detalles importantes, incluidos el número y la complejidad de los elementos de procesamiento, la estructura de la red de interconexión entre los elementos de procesamiento, la coordinación entre los elementos de procesamiento, así como las características importantes del problema a resolver.

Para una investigación más detallada, es útil hacer una clasificación según las características importantes de una computadora paralela. Un modelo simple para tal clasificación está dado por la *taxonomía de Flynn* [15]. Esta taxonomía caracteriza las computadoras paralelas de acuerdo con el control global y los flujos de datos y control resultantes, dividiéndola en 4 categorías [16].

SISD

Una Secuencia de instrucciones y una secuencia de datos. Es un único procesador que interpreta una única secuencia de instrucciones para operar con los datos almacenados en una única memoria, no explota el paralelismo en las instrucciones ni en flujos de datos. Por lo tanto, SISD es la computadora secuencial convencional según *el modelo de von Neumann* [17].

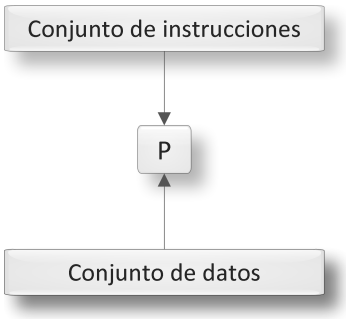


Figura 2.4: SISD

SIMD

Una secuencia de instrucciones y múltiples secuencias de datos. Una sola instrucción controla paso a paso la ejecución simultanea y sincronizada de un cierto número de elementos de procesamiento. Cada uno de ellos tiene una memoria asociada, un computador que explota varios flujos de datos dentro de un único flujo de instrucciones para realizar operaciones que pueden ser paralelizadas de manera natural. Un ejemplo de estas arquitecturas son los procesadores vectoriales.

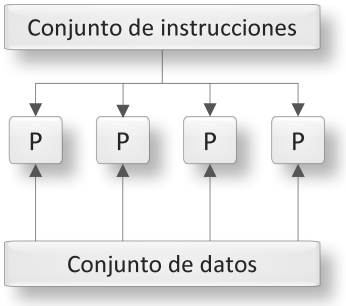


Figura 2.5: SIMD

MISD

Múltiples secuencias de instrucción y una secuencia de datos. Se transmite una secuencia de datos a un conjunto de procesadores, y cada uno de los cuales ejecuta una secuencia de instrucciones diferente. Poco común debido al hecho de que la efectividad de los múltiples flujos de instrucciones suele precisar de múltiples flujos de datos. Se han propuesto algunas arquitecturas teóricas, pero ninguna llegó a producirse en masa.

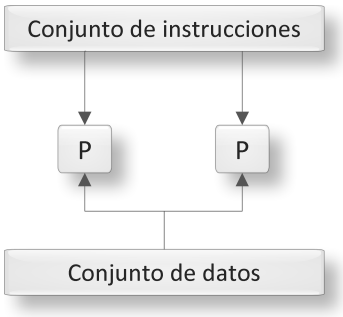


Figura 2.6: MISD

MIMD

Múltiples secuencias de instrucción y múltiples secuencias de datos. Consiste de un conjunto de procesadores que ejecutan simultáneamente secuencias de instrucción diferentes con conjuntos de datos diferentes. Los sistemas multiprocesador y los clusters son ejemplos de esta categoría.

Figura 2.7: MIMD

2.3.5. Multicores y clusters

Multicores

A principios de la década del 2000, el proceso tradicional de mejora de los procesadores llegó a su límite debido a la dicultad de extraer más paralelismo a nivel de instrucciones (ILP, por sus siglas en inglés) de los programas junto al excesivo consumo de potencia y generación de calor que alcanzaban los procesadores. Por tanto, se tuvieron que idear nuevas alternativas a la hora de diseñar procesadores que permitieran incrementar el rendimiento de los mismos. Así es como nacen los procesadores multicores, los cuales están formados por 2 o más núcleos de procesamiento en el mismo chip. Aunque estos núcleos son más limitados y lentos en comparación a sus

antecesores mononúcleo, su combinación mejora tanto el rendimiento global como la eficiencia energética del procesador.

Desde su surgimiento, grandes fabricantes de estas arquitecturas como Intel y AMD, han ido mejorando el diseño de los procesadores multicores a lo largo de las sucesivas familias de procesadores de propósito general lanzadas al mercado. Al principio contenían en una misma oblea dos procesadores mononúcleo. Más tarde, en las siguientes generaciones, han ido aumentando la cantidad de núcleos e incorporando cachés a varios niveles (L2 y L3), las cuales son compartidas por todos los núcleos o parte de ellos [18].

- **Procesadores Intel:** en la actualidad, Intel ofrece una amplia gama de procesadores, cada uno orientado a diferentes usos. A continuación se presentan la familia de procesadores Intel y sus características principales:
  - **Atom** bajo consumo. Utilizados para dispositivos portátiles o aquellos donde el rendimiento energético es crítico.
  - **Celeron** son los más económicos. Utilizados en computadoras de escritorio o portables. Poseen menos núcleos y memorias cache más pequeñas.
  - **Core** mayor poder de cómputo, pensados para uso personal y profesional. Los primeros de la linea en integrar cache L3.
    - **i3** los más modestos de la familia. La mayoría de sus modelos poseen la tecnología Hyper-Threading y memorias cache L3 hasta 8MB.
    - **i5** más potencia que los i3. La mayoría cuentan con 4 núcleos físicos, aunque los hay de 2 núcleos con Hyper-Threading. Cache L3 hasta 6MB y tecnología Turbo Boost.
    - **i7** los más potentes. Los hay de 2, 4 y 6 núcleos, todos con tecnología Hyper-Threading y Turbo Boost. Caches L3 de hasta 20MB.
  - **Xeon** los más potentes que fabrica Intel. Utilizados para workstation y servidores. Soporta múltiples sockets. Mayor número de núcleos (hasta 28 núcleos con Hyper-Threading). Memorias caché L3 más grandes (hasta 60MB). Soporte para detección y corrección de errores en la memoria RAM.
- **Procesadores AMD:** al igual que Intel, AMD ofrece una amplia gama de procesadores diseñados para diferentes propósitos. AMD ofrece 5 líneas principales de procesadores:

- **Athlon** ofrece un buen cociente precio-rendimiento y puede ser una buena opción para una computadora de uso personal. Se ofrecen en configuraciones, de 4 núcleos con memoria caché L2 de 4 Mb.
- **FX** son la gama media actual para el segmento de escritorio y sus modelos poseen 4 núcleos y memoria caché L3 de hasta 4Mb.
- **Ryzen** los actuales tope de gama de AMD. Cuentan con versiones que van desde los 4 núcleos y 4 hilos hasta los 8 núcleos y 16 hilos. Cache L3 hasta 32Mb.
- **Opteron** está destinada a los servidores y workstations, se los puede encontrar en versiones de 4, 6 y 8 núcleos. Estos potentes procesadores integran hasta 16 núcleos y 16Mb de memoria caché L3.
- **EPYC** linea nueva de procesadores para servidores y workstation. Superiores a los Opteron, se los puede encontrar en versiones de 8, 16, 24 y 32 núcleos con 16, 32, 48 y 64 hilos respectivamente. Poseen caché L3 hasta 64Mb.

A continuación se describen las características principales de cada familia AMD:

- **Fusion** corresponde a las APUs que integran CPU y GPU en el mismo chip.
- **Bobcat** está orientada al segmento de bajo consumo y de bajo costo.
- **Bulldozer** fue diseñada para computadoras de escritorio y servidores.
- **Zen** mejora el rendimiento por núcleo más que aumentar su número o la cantidad de hilos hardware.

- **Procesadores ARM:** Pertenecen a la familia de arquitecturas RISC (Reduced Instruction Set Computer) y son desarrollados por la empresa Advanced RISC Machines (ARM).
  ARM fabrica procesadores multicore RISC de 32 y 64 bits. Los procesadores RISC están diseñados para realizar una cantidad menor de tipos de instrucciones de computadora, de modo que puedan funcionar a mayor velocidad y realizar más instrucciones por segundo. Al eliminar las instrucciones innecesarias y optimizar las rutas, los procesadores RISC proporcionan un rendimiento sobresaliente a una fracción de la demanda de energía de los dispositivos

CISC[1] (Complex Instruction Set Computer). Los procesadores ARM se usan ampliamente en dispositivos electrónicos de consumo como teléfonos inteligentes, tabletas, reproductores multimedia y otros dispositivos móviles, como los weareables[2]. Debido a su reducido conjunto de instrucciones, requieren menos transistores, lo que permite un tamaño de dado más pequeño para los circuitos integrados.

El diseño simplificado de los procesadores ARM permite un procesamiento multicore más eficiente y una codificación más sencilla para los desarrolladores. Si bien no tienen el mismo rendimiento de procesamiento que los productos de las marcas líderes, los procesadores ARM tienen un mejor rendimiento energético.

Existen diferentes familias ARM orientadas a distintos usos, los Cortex-R utilizados en procesadores de tiempo real; los Cortex-M, diseñados principalmente para dispositivos móviles por su bajo consumo; los SecurCore diseñado para dispositivos de seguridad; y finalmente la familia Cortex-A que es la utilizada en las SBC.

Aquellos procesadores basados en Cortex-A comparten una arquitectura y conjunto de características comúnmente admitidas. Los procesadores ARMv7-A admiten un conjunto de instrucciones de 32 bits y una ruta de datos, así como el conjunto de instrucciones Thumb2 de 16/32 bits. Los procesadores ARMv8-A Cortex-A añaden soporte para los estados de ejecución AArch32 / AArch64. Los procesadores Cortex-A de ARMv8.2-A agregan extensiones de arquitectura para un mayor impacto en el rendimiento del sistema. Todos los procesadores Cortex-A son compatibles con versiones anteriores y están respaldados por un fuerte ecosistema ARM.

Obviando las variantes que han quedado obsoletas, actualmente podemos encontrar las siguientes microarquitecturas:

- **ARMv7-A** estamos ante un modelo de 32 bits que ha servido para crear soluciones muy potentes. Con el salto a los 64 bits han quedado relegadas

[1]Los microprocesadores CISC (Complex Instruction Set Computer, del español Computadora con conjunto de instrucciones complejas) tienen un conjunto de instrucciones que se caracteriza por ser muy amplio y permitir operaciones complejas entre operandos situados en la memoria o en los registros internos, en contraposición a la arquitectura RISC.

[2]Wearable hace referencia al conjunto de aparatos y dispositivos electrónicos que se incorporan en alguna parte de nuestro cuerpo interactuando de forma continua con el usuario y con otros dispositivos con la finalidad de realizar alguna función concreta. Ejemplos de estos son: relojes inteligentes o smartwatchs, zapatillas de deportes con GPS, etc.

a un segundo plano y se mantienen sobre todo en productos donde lo que prima es el bajo consumo. Algunos de los procesadores que han basado en esta microarquitectura son: Cortex-A8 utilizado en los Apple iPod touch (tercera generación), y Cortex-A9, el cual puede ser encontrado en los celulares Galaxy Nexus.

- **ARMv8-A** primeras unidades de 64 bits de la arquitectura y la más utilizada actualmente, ya que integra soluciones versátiles que permiten crear sistemas en chip (SoCs, por sus siglas en inglés) equilibrados tanto en consumo como en rendimiento, sobre todo gracias a la combinación de diferentes variantes en configuraciones asimétricas (big.LITTLE). Tres de sus variantes más importantes son:
  - **Cortex-A53** son núcleos de bajo consumo que ofrecen un buen nivel de rendimiento, pero se encuentran por debajo de las otras dos variantes mencionadas a continuación. Se utilizan sobre todo en smartphones y tablets económicos y como segundo módulo de bajo consumo en CPUs con estructura big.LITTLE.
  - **Cortex-A57** son soluciones de alto rendimiento, aunque su consumo es mayor que el de los Cortex-A53. Actualmente se encuentran en productos de gama alta, aunque suelen ir combinados normalmente con los núcleos Cortex-A53 para mejorar la eficiencia conjunta del SoC en el que se integran.
  - **Cortex-A72** es una importante revisión que evoluciona desde los núcleos Cortex-A57. Mejoran el rendimiento, el consumo y permitiendo diseños más pequeños al ocupar una menor superficie dentro del encapsulado. No marcan una mejora sustancial en rendimiento, pero sí en consumo y reducción de costos.
- **ARMv8.2-A** es una evolución de la arquitectura ARMv8-A, la cual incluye cambios como un modelo de memoria mejorado, procesamiento de datos de punto flotante de precisión media e introduce el soporte RAS (Reliability, Availability and Serviceability). Ejemplos de procesadores de esta arquitectura son el Cortex-A55 y el Cortex-A75.

Clusters

Un cluster es un tipo de sistema de procesamiento paralelo compuesto por un conjunto de componentes de hardware estándares interconectadas vía algún tipo de

red, las cuales cooperan configurando un recurso que se ve como "único e integrado", más allá de la distribución física de sus componentes. Cada uno de los componentes que conforman un cluster se denomina nodo y son los encargados de llevar adelante el procesamiento. La construcción de los nodos de un clúster es relativamente fácil y económica debido a su flexibilidad: pueden tener todos la misma configuración de hardware y sistema operativo (clúster homogéneo), o tener diferente hardware y/o sistema operativo (clúster heterogéneo). Esta característica constituye un elemento importante en el análisis del rendimiento que se puede obtener de un clúster como máquina paralela.

Para que los nodos se comuniquen entre sí, es necesario proveerlos de un medio de interconexión mediante algún tipo de red de alta velocidad, por ejemplo, una red LAN. Sin embargo, no basta sólo con interconectar nodos para que un cluster funcione como tal, sino que es necesario proveer al mismo de un sistema de administración de cluster, el cual se encargue de interactuar con el usuario y las aplicaciones que se ejecuten en él.

Éste tipo de sistemas ofrece una manera rentable de mejorar el rendimiento (velocidad, disponibilidad, rendimiento, etc.) comparado con supercomputadoras de similares características [19]. A continuación se detallan los tipos de cluster según sus características.

- Cluster de alto rendimiento (High Performance Clusters): se caracterizan por ejecutar tareas que requieren de gran capacidad computacional, grandes cantidades de memoria, o ambos a la vez. El realizar estas tareas puede comprometer los recursos del cluster por periodos de tiempo indeterminados.

- Cluster de alta disponibilidad (High Availability): el objetivo principal de estos cluster se centra en la disponibilidad y la confiabilidad. Los clusters que pertenecen a estar categoría intentan brindar la máxima disponibilidad de los servicios que ofrecen. La confiabilidad es provista mediante software que detecta fallos y permite recuperarse frente a los mismos, mientras que por medio de hardware se previene tener un único punto de fallos.

- Cluster de alta eficiencia (High Throughout): son clusters cuyo objetivo de diseño se centra en ejecutar la mayor cantidad de tareas en el menor tiempo posible. Para poder llevar adelante esto debe existir independencia de datos entre las tareas individuales. El retardo entre los nodos del cluster no es considerado un gran problema.

Si bien cada tipo de cluster visto tiene sus características representativas, todos tienen un objetivo común: obtener un alto rendimiento a un bajo costo. Si a esto se le suma que pueden escalarse fácilmente, esto explica porqué el uso de clusters es hoy una de las posibilidades de cómputo paralelo/distribuido más elegidas.

2.4. Modelos y librerías de programación

Diversos lenguajes de programación y librerías han sido desarrollados para la programación paralela explícita. Estas difieren principalmente en la manera en que el usuario ve al espacio de direcciones. Los modelos se dividen básicamente en los que proveen un espacio de direcciones compartido o uno distribuido, aunque también existen modelos híbridos que combinan las características de ambos (este último modelo surge con la aparición de los procesadores multicore).

El espacio de direcciones influye significativamente sobre la manera en que los hilos o procesos intercambian la información. A continuación se describen cada uno de los modelos [11]

2.4.1. Paradigma de pasajes de mensajes

Cuando el espacio de direcciones es distribuido, cada proceso tiene su memoria local, y no existe una memoria compartida a la cual todos los procesos puedan acceder para intercambiar información. Es por ello que el intercambio de información entre procesos se da enviando y recibiendo mensajes.

Principios del paradigma de pasaje de mensajes

Como se mencionó anteriormente, en este paradigma varios procesos se ejecutan en paralelo y se comunican enviando y recibiendo mensajes. Los mismos operan sobre espacios de direcciones disjuntos, es decir, no tienen acceso a una memoria compartida. Toda la comunicación que pueda haber entre ellos se lleva adelante por intercambio de mensajes. Las operaciones básicas de comunicación son *send* y *receive*.

Existen diferentes protocolos para las operaciones *send* y *receive*: bloqueantes y no bloqueantes. Los protocolos bloqueantes no devuelven el control de la operación hasta que el dato a transmitir esté seguro. Si bien garantizan la corrección de las operaciones, pueden incurrir en overheads innecesarios por la ocurrencia de ocio, además de ser más propensos a ocasionar deadlocks. En sentido opuesto, los protocolos no bloqueantes devuelven el control inmediatamente luego de iniciar la operación *send/recv*, por lo

que el overhead asociado es mínimo. Sin embargo, la corrección de sus operaciones queda a cargo del programador.

MPI

La especificación MPI (Message Passing Interface) define un estándar para el pasaje de mensajes que puede ser utilizado desde los lenguajes C o Fortran, y potencialmente desde otros también. MPI define tanto la sintaxis como la semántica de un conjunto básico de rutinas, las cuales resultan útiles a la hora de escribir programas con mensajes. En la actualidad, existen diversas implementaciones para diferentes proveedores de hardware [20].

Todas las rutinas, tipos de datos y constantes en MPI tienen el prefijo "MPI_". El código de retorno para operaciones terminadas exitosamente es MPI_SUCCESS. Aunque MPI posee cientos de funciones, básicamente con 6 rutinas podemos escribir programas paralelos basados en pasaje de mensajes: MPI_Init, MPI_Finalize, MPI_Comm_size, MPI_Comm_rank, MPI_Send y MPI_Recv.

Para poder utilizar las rutinas provistas por MPI es necesario que todos los procesos invoquen la operación MPI_Init. La misma inicializa el ambiente MPI. La rutina MPI_Finalize debe ser llamada al finalizar la computación, ya que se encarga de realizar diferentes tareas de mantenimiento para poder cerrar el ambiente MPI.

También existen rutinas que proveen información del dominio de comunicación y de todos los procesos que están involucrados en la ejecución de una aplicación, las funciones que permiten adquirir esta información son: *MPI_Comm_size*, que indica la cantidad de procesos en el dominio de comunicación, y *MPI_Comm_rank* que indica el "rank" (identificador) del proceso dentro de un dominio de comunicación.

MPI también ofrece diferentes rutinas que implementan las operaciones básicas *send* y *receive*. Para comunicación bloqueante, ofrece una primitiva de recepción como:

- *MPI_Recv* rutina básica para recibir datos de otro proceso. La operación se completa sólo después de que los datos hayan sido recibidos y copiados.
- *MPI_Send* rutina básica para enviar datos a otro proceso. La operación se completa cuando el mensaje es recibido.

Al igual que para la comunicación bloqueante, por medio de las primitivas *MPI_Send* y MPI_Recv, MPI provee primitivas de recepción y emisión para comunicación no bloqueante. Las primitivas son *MPI_Irecv* y *MPI_Isend*. Además, para poder chequear la finalización de las operaciones send y receive no bloqueantes, MPI provee

dos funciones: *MPI_ Test* y *MPI_ Wait*. La primera chequea si una operación ha finalizado o no, y la segunda permite bloquear al proceso hasta que una operación no bloqueante realmente finalice.

MPI provee un conjunto de funciones que realizan operaciones de comunicación colectivas comúnmente usadas. Todas estas funciones toman como argumento un comunicador que define el grupo de procesos involucrados en la operación de comunicación. Todos los procesos pertenecientes a este comunicador participan en la operación y es necesario que todos ellos invoquen a la función correspondiente. Siempre que sea posible, conviene emplear estas operaciones para comunicaciones grupales dado que se encuentran optimizadas a nivel de librería. Entre las más usadas se encuentran:

- *MPI_ Bcast* Envía un mensaje a todos los procesos restantes.
- *MPI_ Scatter* Separa y distribuye datos entre todos los procesos.
- *MPI_ Gather* Recibe y concatena los datos enviados por todos los procesos restantes
- *MPI_ Reduce* Combina mensajes de todos los procesos.

2.4.2. Paradigma de memoria compartida

Cuando el espacio de direcciones es compartido, el proceso y sus hilos acceden al mismo espacio de memoria, y estos intercambian información leyendo y escribiendo sobre variables que son compartidas.

Hilos

En el modelo de hilos, cada proceso puede incluir múltiples flujos de control independientes los cuales son llamados hilos.

Un atributo característico es que los hilos de un proceso comparten el espacio de direccionamiento del mismo, es decir, tienen un espacio común. Cuando un hilo almacena un valor en el espacio de direcciones compartido, otro del mismo proceso también podrá accederlo [16].

A continuación se mencionan algunas ventajas del modelo de hilos:

- Portabilidad del software: las aplicaciones multi-hiladas pueden ser desarrolladas en máquinas secuenciales y luego ser trasladadas a máquinas paralelas sin tener que realizar cambios.

- Ocultamiento de la latencia: una de las mayores fuentes de overhead, tanto en programas seriales como paralelos, es la latencia en el acceso a memoria, de la E/S y de la comunicación. La ejecución multi-hilada ayuda a ocultar la latencia, al poder intercalar la ejecución de los mismos.

- Planificación y balance de carga: usualmente resulta difícil obtener una distribución del trabajo balanceada en aplicaciones poco estructuradas y dinámicas. Los hilos permiten al programador especificar un gran número de tareas concurrentes que serán mapeadas a unidades de procesamiento de forma dinámica. De esta manera, se libera al programador de la responsabilidad de la planificación explícita y del balance de carga.

- Facilidad de programación y amplio uso: debido a las ventajas mencionadas anteriormente, los programas multi-hilados resultan más fáciles de escribir que los correspondientes programas utilizando pasaje de mensajes. Sin embargo, lograr igual rendimiento para ambos programas puede requerir un esfuerzo mayor.

Pthreads

POSIX Threads (Pthreads) es una API para la creación y manipulación de hilos, que ha emergido como estándar. Existen diversas implementaciones para los distintos sistemas operativos como por ejemplo: GNU/Linux, FreeBSD, NetBSD, OpenBSD, entre otras.

Para poder escribir un programa multihilado, es necesario contar con una función que permita crear hilos. Para ello la API Pthreads provee la función *pthread_ create*.

En muchas ocasiones un hilo debe esperar a que otro termine para poder realizar su trabajo. La función *pthread_ join* suspende la ejecución del hilo invocador hasta que el hilo especificado como parámetro finalice con su trabajo.

Cuando múltiples hilos intentan manipular el mismo dato, los resultados pueden ser incoherentes si no se toman los cuidados apropiados. Los programadores deben garantizar que las diferentes tareas concurrentes accedan a los datos en forma sincronizada de manera de obtener programas hilados que sean correctos. Pthreads provee soporte para implementar secciones críticas y operaciones atómicas usando *mutex-locks*. Asimismo, ofrece variables *condition* para los casos en que los hilos deben sincronizar por condición.

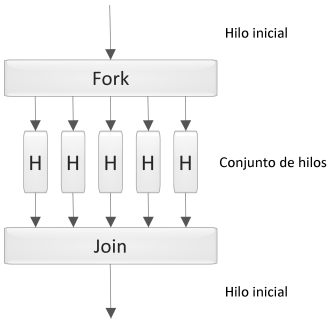


Figura 2.8: Modelo Fork-Join de OpenMP.

OpenMP

OpenMP es una API que reúne una colección de directivas, librerías y variables de entorno para la programación de aplicaciones paralelas sobre arquitecturas con memoria compartida, disponible para los lenguajes C, C++ y Fortran. Es una herramienta multiplataforma ya que posee versiones para diversas arquitecturas y sistemas operativos, haciendo que las aplicaciones paralelas que la utilizan sean fácilmente portables.

En general, el programa paralelo surge a partir de añadir simplemente directivas al programa secuencial que permiten: crear un conjunto de hilos, repartir tareas paralelas independientes entre ellos y sincronizarlos para el acceso de datos compartidos.

OpenMP utiliza el modelo *fork-join* (Figura 2.8), en el cual el programa comienza ejecutándose como un único hilo (llamado *maestro*). Cuando se encuentra la primera directiva paralela se crea un conjunto de hilos (cada uno ejecutará el mismo código), los cuales podrán repartirse el trabajo y/o coordinarse para el uso de datos compartidos a través del uso de directivas para tal fin. Al terminar la región paralela continúa la ejecución el hilo maestro.

Los hilos se ejecutarán dentro del mismo espacio de direcciones y podrán compartir el acceso a variables declaradas en el mismo; también se permite que una variable sea designada como privada a un hilo, en este caso cada hilo tendrá una copia que usará mientras dure la región paralela.

2.4.3. Paradigma híbrido

Los paradigmas de programación tradicionales (pasaje de mensajes y memoria compartida) no se adaptan naturalmente a los clusters de multicores. Al mismo tiempo, se espera que un modelo híbrido (combinación de pasaje de mensajes con memoria compartida) explote mejor sus características.

La idea básica es que las tareas que se encuentran en el mismo nodo se comuniquen y sincronicen por memoria compartida mientras que las que se encuentran en diferentes nodos lo hagan por pasaje de mensajes.

En la práctica, se usa generalmente MPI en combinación con OpenMP o Pthreads.

2.5. Diseño de algoritmos y técnicas para optimizar rendimiento

El diseño de aplicaciones paralelas básicamente se resume a dos etapas: descomposición y mapeo.

2.5.1. Etapa de descomposición

Uno de los pasos fundamentales en el desarrollo de algoritmos paralelos es la división del trabajo en un conjunto de tareas que puedan ejecutarse en forma concurrente. No existe una única técnica que permita descomponer un problema. La elección de la misma está supeditada a las características propias del problema y a la elección del diseñador del algoritmo. Se debe tener en cuenta que una descomposición dada no siempre permite alcanzar el mejor algoritmo paralelo para un problema determinado. Esta descomposición puede realizarse de muchos modos. Un primer concepto es pensar en tareas de igual código (normalmente paralelismo de datos o dominio) pero también podemos tener diferente código (paralelismo funcional o tareas o de control).

Descomposición de datos

La descomposición de datos es un método potente y comúnmente utilizado para derivar concurrencia en aquellos algoritmos que trabajan sobre grandes estructuras de datos. En esta técnica, la descomposición se realiza en dos pasos. En el primer paso, se particionan los datos sobre los cuales se opera, mientras que en el segundo paso, se utiliza el particionamiento anterior para inducir una partición del trabajo a realizar en tareas. Por lo general, las tareas realizan el mismo trabajo o trabajos similares sobre las diferentes particiones de datos.

Descomposición funcional

El paralelismo funcional se refiere a la ejecución simultánea de distintos flujos de instrucciones. Estas instrucciones pueden ser aplicadas al mismo o distintos flujos de

datos (típicamente se da el segundo caso). Se realiza una descomposición funcional y las tareas generadas ejecutan independientemente, comunicándose cuando lo necesitan [11].

2.5.2. Etapa de mapeo

El siguiente paso luego de que el problema ha sido descompuesto en tareas, consiste en asociar dichas tareas a las unidades de procesamiento donde serán ejecutadas. A esta asociación se la conoce como mapeo y tiene como objetivo minimizar los overheads ligados a la ejecución paralela de las tareas. Las técnicas de distribución utilizadas en los algoritmos paralelos pueden clasificarse en dos categorías: estático y dinámico. Entre los factores que determinan que técnica emplear se encuentran el paradigma de programación elegido, el tipo de arquitectura, las características de las tareas y la interacción que se da entre estas [11].

Mapeo estático

En las técnicas de mapeo estático, las tareas se distribuyen entre los procesadores antes de comenzar con la ejecución del algoritmo concreto. En general es más fácil de diseñar e implementar, y es más eficiente (si se logra balancear la carga de trabajo); pero requiere que se tenga conocimiento a priori de la cantidad y tamaño de las tareas a realizar.

Mapeo dinámico

El mapeo dinámico es necesario cuando el estático puede llevar a una distribución de carga de trabajo con un alto grado de desbalance entre los procesadores, o cuando las tareas a realizar se generan dinámicamente durante la ejecución de la aplicación. En las técnicas de mapeo dinámico, el trabajo (datos o tareas) se distribuye entre los procesadores durante la ejecución del algoritmo. Esta clase de mapeo es la mejor opción cuando la cantidad o tamaño de las tareas no son conocidos a priori [11].

2.5.3. Métodos para reducir overhead

El overhead generado por la interacción entre procesos limita la eficiencia que un programa paralelo puede alcanzar. Son varios los factores que inciden sobre el overhead generado por la interacción: el volumen de datos intercambiados, la frecuencia de la interacción, etc.

Entre los métodos para reducir overhead, se encuentran: maximización de la localidad de datos para combatir las limitaciones del sistema de memoria, minimización de la competencia de recursos para evitar interacciones innecesarias, solapamiento de cómputo con comunicaciones para reducir tiempos ociosos, replicación de datos o cómputo para evitar comunicaciones, uso de operaciones de comunicación colectivas, entre otros.

2.6. Modelo de algoritmos paralelos

Un modelo de algoritmo permite estructurar la computación de un algoritmo paralelo. Es la combinación de las técnicas de descomposición y mapeo elegidas junto a una estrategia para reducir overhead. Según [11], existen diferentes modelos de algoritmos paralelos, como el SPMD (Single Program Multiple Data), grafo de tareas, productor-consumidor o pipeline, master/slave, entre otros. Se describen a continuación los usados en esta tesina.

2.6.1. Modelo SPMD

En el modelo SPMD (Single Program Multiple Data) se genera un programa único que cada nodo ejecuta sobre una porción diferente del dominio de datos. Por medio de instrucciones en el código se puede hacer que cada nodo tome distintos caminos del programa.

La implementación de un programa que siga este modelo involucra dos grandes fases: elección de la distribución de datos y generación del programa paralelo [21]. En la etapa de elección de la distribución de datos se determina que nodo procesa cada dato. Una mala distribución de los datos podría repercutir directamente sobre la carga de trabajo y sobre la cantidad de comunicaciones generadas.

2.6.2. Modelo Master-Slave

En el modelo Master/Slave (o también conocido como Master/Worker), un proceso actúa como administrador (master) coordinando a todos los demás procesos que funcionan como esclavos (workers). El master es el encargado de descomponer el problema en tareas pequeñas, distribuirlas entre los diferentes procesos worker, y recibir los resultados parciales para componer la solución final del problema. Los workers realizan un procesamiento muy simple: reciben una tarea a realizar, la procesan, y

envían los resultados al master. En ocasiones, el master también computa en simultáneo con los workers. Este ciclo puede repetirse hasta que el master no tenga más tareas para resolver [11].

2.7. Métricas de rendimiento

Es importante estudiar el rendimiento de programas paralelos con el fin de determinar el mejor algoritmo, evaluar plataformas de hardware, y examinar los beneficios del paralelismo. Para ello se introduce una serie de métricas que permitirán realizar el análisis deseado.

2.7.1. Tiempo de ejecución

El tiempo de ejecución secuencial corresponde al tiempo de ejecución de un programa secuencial que por lo tanto emplea una única unidad de procesamiento. El tiempo de ejecución paralelo es el tiempo que transcurre desde el momento en que el procesamiento paralelo comienza hasta que el último proceso finaliza su ejecución. Se expresa el tiempo de ejecución secuencial como T_s y el tiempo de ejecución paralelo como T_p [11].

2.7.2. Speedup

El speedup refleja el beneficio relativo de paralelizar la solución a un problema. Se define como el cociente entre el tiempo requerido por la solución secuencial utilizando la unidad de procesamiento más potente de la arquitectura paralela y el tiempo requerido por la solución paralela de interés utilizando p unidades de procesamiento. A continuación se ilustra la ecuación que permite calcular esta métrica.

$$S = \frac{T_s}{T_p} \tag{2.1}$$

Específicamente, el speedup indica cuántas veces más rápido se obtiene la solución al problema utilizando la solución paralela con p unidades de procesamiento con respecto a la solución secuencial.

Cuando hacemos referencia a la solución secuencial se debe tener en cuenta que puede existir más de un algoritmo que resuelva el problema. Se debe elegir el que soluciona el problema en la menor cantidad de tiempo. De otra manera no es justa la comparación con el algoritmo paralelo [11].

De acuerdo al resultado obtenido de la Ecuación 2.1 se pueden obtener tres tipos de speedup:

Speedup lineal también llamado speedup perfecto debido a que el algoritmo paralelo se ejecuta p veces más rápido que el algoritmo secuencial.

Speedup sublineal es el caso más frecuente en el cual el speedup obtenido es menor que p debido a factores como comunicación, sincronización, etc.

Speedup superlineal es el caso menos frecuente, en el cual el speedup obtenido supera a p. Esta situación puede darse por 2 motivos. El primero es porque el algoritmo paralelo realiza menos trabajo que la versión secuencial, como es el caso de los recorridos en árboles o grafos. El segundo es por una mejora en la localidad de datos al distribuir los mismos, dejando a la versión secuencial en desventaja [9].

2.7.3. Eficiencia

La eficiencia es una medida que refleja cuan bien utilizadas son las unidades de procesamiento para resolver el problema. En arquitecturas homogéneas la eficiencia (E) se define como el cociente entre el speedup (S) y la cantidad de elementos de procesamiento usados (p), como se indica en la ecuación 2.2.

$$E = \frac{S}{p} \tag{2.2}$$

Se puede obtener una eficiencia igual a 1 si el speedup obtenido con p unidades de procesamiento es igual a p. Generalmente, debido a los costos que implican la sincronización y la comunicación, el speedup es menor a p, lo que lleva a la eficiencia a ser menor a 1.

2.7.4. Escalabilidad

En muchos casos las aplicaciones paralelas son diseñadas y probadas para pequeños problemas y/o en un número reducido de elementos de procesamiento, a pesar de que se deban utilizar en problemas y arquitecturas mucho mayores. Mientras que el desarrollo y prueba de la solución paralela se simplifica al utilizar instancias reducidas del problema y de la arquitectura, el rendimiento es muy difícil de predecir en forma precisa basado en esos resultados. Esto se debe a que los algoritmos pueden tener

distinto comportamiento (referido al rendimiento) para los diferentes tamaños de problema y dimensiones de la arquitectura utilizada.

La escalabilidad de un sistema paralelo es una medida de la capacidad de incrementar el speedup en proporción a la cantidad de procesadores de la arquitectura, reflejando la habilidad de utilizar efectivamente el incremento de los recursos de procesamiento. Su análisis es necesario para elegir la mejor combinación de algoritmo y arquitectura para resolver un problema bajo distintas restricciones sobre el crecimiento del tamaño del problema o de los recursos de procesamiento

2.7.5. MIPS y FLOPS

Una medida de rendimiento que a veces se usa en la práctica para evaluar un sistema informático es la tasa de millones de instrucciones por segundo (MIPS). La tasa de MIPS de A esta definida como:

$$MIPS(A) = \frac{Instrucciones(A)}{T(A) \times 10^6}$$

donde instrucciones(A) es el número de instrucciones ejecutadas por el programa A y T(A) es el tiempo de ejecución de dicho programa.

Usar las tasas de MIPS como medida de rendimiento tiene algunos inconvenientes. Primero, la tasa de MIPS solo considera el número de instrucciones, lo que favorece a los procesadores con instrucciones simples sobre los procesadores con instrucciones más complejas. En segundo lugar, la tasa MIPS de un programa no se corresponde necesariamente con su tiempo de ejecución: al comparar dos programas A y B en un procesador X, B puede tener una tasa de MIPS más alta que A, pero A tiene un tiempo de ejecución más pequeño [16].

Para programas con cálculos científicos, a veces se utiliza la velocidad MFLOPS (Millones de operaciones de coma flotante por segundo). La tasa de MFLOPS de un programa A está definida por

$$MFLOPS(A) = \frac{\text{instrucciones\_pf}(A)}{T(A) \times 10^6}$$

,

donde instrucciones_pf(A) es el número de operaciones de coma flotante ejecutadas por A. La tasa MFLOPS no se basa en el número de instrucciones ejecutadas, como es el caso de la tasa MIPS, sino en el número de operaciones aritméticas en

valores de coma flotante realizado por la ejecución de sus instrucciones. Las instrucciones que no realizan operaciones de punto flotante no tienen efecto en la velocidad MFLOPS. Dado que se usa la cantidad efectiva de operaciones realizadas, la tasa de MFLOPS proporciona una comparación justa de las diferentes versiones de programa que realizan las mismas operaciones, y las mayores tasas de MFLOPS corresponden a tiempos de ejecución más rápidos.

Una desventaja de utilizar la tasa MFLOPS como medida de rendimiento es que no hay diferenciación entre los diferentes tipos de operaciones de coma flotante realizadas. Por lo tanto, los programas con operaciones de coma flotante más simples son preferibles a los programas con operaciones más complejas. Sin embargo, la tasa MFLOPS es adecuada para comparar versiones de programa que realizan las mismas operaciones de coma flotante [16].

2.7.6. Rendimiento/Watt

El consumo energético se ha vuelto uno de los mayores desafíos en el campo de la computación. Es por eso que se necesita contar con métricas con las cuales sea posible medir o estimar la eficiencia energética. Una medida aceptada es el número de FLOPS por Watt requerido (FLOPS/W) [22].

2.8. Resumen

La computación paralela se origina por las limitaciones de los computadores secuenciales: integrando varios procesadores para llevar a cabo la computación es posible resolver problemas que requieren de más memoria o de mayor velocidad de cómputo. Su objetivo principal es reducir el tiempo de resolución de problemas computacionales, o bien para resolver problemas grandes que no podrían ser resueltos por un computador convencional. Los problemas habituales en los cuales se aplica la programación paralela son: problemas con alta demanda de cómputo, problemas que requieren procesar una gran cantidad de datos, o problemas de tiempo real, en los que se necesita la respuesta en un tiempo máximo. Sin embargo, en las últimas décadas la eficiencia energética ha cobrado un valor semejante. Esto se debe al elevado consumo energético y generación de calor de las arquitecturas paralelas que afectan al funcionamiento y repercuten en el costo, debido a la adquisición de equipos para refrigeración y al consumo energético que generan los mismos.

Ahora bien, desarrollar soluciones paralelas no es tarea sencilla, a lo que pueden

presentarse dificultades. Existen diversas dicultades que se pueden encontrar a la hora de escribir un programa paralelo. Por ejemplo, no siempre es posible paralelizar un programa; la paralelización requiere de tareas que no están presentes en la programación secuencial (descomposición del problema, comunicación y sincronización, mapeo, entre otras); mayor propensión a cometer errores por aumento de la complejidad; mayor dificultad a la hora de probar o depurar un programa; y por último, la fuerte dependencia entre el programa paralelo y la arquitectura de soporte para obtener alto rendimiento.

Por otro lado, también es importante conocer la plataforma donde se aplicará la solución. Así como se pueden clasificar las dificultades en físicas y lógicas, también se lo puede hacer con las plataformas paralelas. Se entiende como organización lógica, a la manera en que el programador visualiza la plataforma paralela, por otro lado, la organización física se refiere al hardware real de la plataforma; se pueden identificar: plataforma de memoria compartida (dos o más unidades de procesamiento conectadas a múltiples módulos de memoria, interconectados entre si por una red) y plataformas de memoria distribuida (p nodos de procesamiento independientes con módulos de memoria locales). En la actualidad, con la incorporación de los procesadores multicore a las arquitecturas de clusters tradicionales se puede identificar un tercer tipo: las plataformas híbridas.

A principios de la década del 2000, el proceso tradicional de mejora de los procesadores llegó a su límite debido a la dicultad de extraer más paralelismo a nivel de instrucciones (ILP, por sus siglas en inglés) de los programas junto al excesivo consumo de potencia y generación de calor que alcanzaban los procesadores. Por tanto, se tuvieron que idear nuevas alternativas a la hora de diseñar procesadores que permitieran incrementar el rendimiento de los mismos. Así es como nacen los procesadores multicores, los cuales están formados por 2 o más núcleos de procesamiento en el mismo chip. Auqnue estos núcleos son más limitados y lentos en comparación a sus antecesores mononúcleo, su combinación mejora tanto el rendimiento global como la eficiencia energética del procesador.

En la actualidad es posible encontrar grandes fabricantes de estas arquitecturas, tales como Intel, AMD, y ARM. Cada uno ofreciendo una amplia variedad de familias.

Un cluster es un tipo de sistema de procesamiento paralelo compuesto por un conjunto de componentes de hardware estándares interconectadas vía algún tipo de red, las cuales cooperan configurando un recurso que se ve como "único e integrado", más allá de la distribución física de sus componentes. Según sus características un cluster puede ser catalogado como: cluster de alto rendimiento (High Performance Clusters),

cluster de alta disponibilidad (High Availability) o cluster de alta eficiencia (High Throughout). Si bien cada tipo de cluster tiene sus características representativas todos tienen un objetivo común, obtener un alto rendimiento a un bajo costo. Si a esto se le suma que pueden escalarse fácilmente, esto explica porqué el uso de clusters es hoy una de las posibilidades de cómputo paralelo/distribuido más elegidas.

A la hora de programar es importante conocer sobre que espacio de direcciones se está trabajando. Los modelos se dividen básicamente en los que proveen un espacio de direcciones compartido o uno distribuido, aunque también existen modelos híbridos que combinan las características de ambos (este último modelo surge con la aparición de los procesadores multicore).

Cuando el espacio de direcciones es distribuido, cada proceso tiene su memoria local, y no existe una memoria compartida a la cual todos los procesos puedan acceder para intercambiar información. Es por ello que el intercambio de información entre procesos se da enviando y recibiendo mensajes. La especificación MPI (Message Passing Interface) define un estándar para el pasaje de mensajes que puede ser utilizado desde los lenguajes C o Fortran, y potencialmente desde otros también. MPI define tanto la sintaxis como la semántica de un conjunto básico de rutinas, las cuales resultan útiles a la hora de escribir programas con mensajes.

Por otro lado, cuando el espacio de direcciones es compartido, el proceso y sus hilos acceden al mismo espacio de memoria, y estos intercambian información leyendo y escribiendo sobre variables que son compartidas. En el modelo de hilos, cada proceso puede incluir múltiples flujos de control independientes los cuales son llamados hilos. Un atributo característico es que los hilos de un proceso comparten el espacio de direccionamiento del mismo, es decir, tienen un espacio común. PThreads y OpenMP son ejemplos de herramientas para la programación sobre memoria compartida; ambas son APIs que facilitan la programación con hilos en múltiples sistemas operativos.

Por último, la idea básica de un modelo híbrido de programación OpenMP/MPI consiste en permitir que cualquier proceso MPI genere un conjunto de hilos OpenMP de la misma manera que lo hace el hilo principal en un programa OpenMP puro, y de esta forma se aprovechan las características de ambos paradigmas.

Diseñar una solución paralalela puede resumirse en dos etapas: etapa de descomposición y etapa de mapeo. En la primera, se basa principalmente en la división del trabajo en un conjunto de tareas que puedan ejecutarse en forma concurrente. Es importante tener en cuenta, que no existe una única técnica que permita descomponer un problema; ejemplos de estas técnicas pueden ser: la descomposición de datos y la descomposición funcional. El siguiente paso luego de que el problema haya sido

descompuesto en tareas es continuar con la etapa de mapeo que consiste en asociar dichas tareas a las unidades de procesamiento donde serán ejecutadas. Las técnicas de mapeo utilizadas en los algoritmos paralelos pueden clasificarse en dos categorías: estático y dinámico.

Entre los métodos para reducir overhead, se encuentran: maximización de la localidad de datos para combatir las limitaciones del sistema de memoria, minimización de la competencia de recursos para evitar interacciones innecesarias, solapamiento de cómputo con comunicaciones para reducir tiempos ociosos, replicación de datos o cómputo para evitar comunicaciones, uso de operaciones de comunicación colectivas, entre otros.

De la combinación de las técnicas de descomposición y mapeo elegidas junto a una estrategia para reducir overhead surgen los diferentes modelos de algoritmos los cuales permiten estructurar la computación de un algoritmo paralelo. Los utilizados en esta tesina son: el modelo SPMD y el modelo Master-Slave.

Es importante estudiar el rendimiento de programas paralelos con el fin de determinar el mejor algoritmo, evaluar plataformas de hardware, y examinar los beneficios del paralelismo. Para ello se han descrito una serie de métricas que permiten realizar el análisis deseado. Para evaluar rendimiento se cuenta con el tiempo de ejecución, el speedup, la eficiencia, la escalabilidad, y las tasas de MIPS y FLOPS. En cuenta a eficienca energética, se suele usar FLOPS/W.

Capítulo 3

Raspberry Pi

En este capítulo se presenta a la familia de placas Raspberry Pi, describiendo sus diferentes modelos, características y capacidades.

3.1. Definiciones

La Raspberry Pi (RPi) es un SBC (Single Board Computer) que tiene el tamaño de una tarjeta de crédito, desarrollada por la fundación Raspberry Pi en Reino Unido. La RPi se ha hecho masiva por su bajo costo, su pequeño tamaño que permite soluciones portables y la posibilidad de incorporar una amplia variedad de sensores, lo que posibilita diversificar sus dominios de aplicación.

3.2. Historia

La idea de la computadora RPi surge en el año 2006. En esa época, Eben Upton trabajaba en el Cambridge Computer Laboratory de Inglaterra. Junto a varios de sus colegas del laboratorio, Eben notó que, en los últimos años, los ingresantes a la carrera de Ciencias de la Computación tenían pocos conocimientos del funcionamiento interno de una computadora.

Surge entonces la idea de recrear el ecosistema de las microcomputadoras de los 80s que permitían una comprensión total del hardware/software y fomentaban la programación y modificación de la computadora, en vez del simple uso del software. El retroceso de la computación como ciencia en las escuelas estaba causando problemas en el ámbito universitario y había que hacer un esfuerzo para revertir la situación.

En 2006 se propone una computadora de 25USD hecha con un microcontrolador

ATmega644 con 512Kb de RAM y salida de vídeo. Aunque Eben Upton deja la universidad y empieza a trabajar en Broadcom, la idea no desaparece, sino que muta con la aparición de SoCs cada vez más poderosos y de precio accesible que permiten el uso de computadoras a jóvenes con intereses más allá de la programación.

En 2012 aparece la RPi 1 portando el BCM2835 (ARMv6Z), un SoC para reproducción multimedia y, aunque de manera lenta, el progreso continúa con la RPi 2 en 2015, utilizando un BCM2836 (ARMv7A). El crecimiento se potencia con el surgimiento de la RPi 3 en 2016, la cual incluye un procesador BCM2837 (ARMv8-A), con la novedad de ser de 64 bits dejando atrás a sus antecesores de 32 bits.

La potencia de cómputo de una RPi 3 es similar a la de un Pentium 4, pero casi dos órdenes de magnitud menor en disipación de calor.

El éxito de las RPi es uno de los motivos de la popularidad de las plataformas ARM. En la actualidad, la mayoría de las distribuciones de GNU/Linux incorporan *cross-compilers*, *cross-assemblers* y *cross-debuggers* para el desarrollo de software en este tipo de plataformas.

3.3. Áreas de aplicación

En cuanto al uso que se le puede dar a la RPi, existen diversos trabajos realizados en los últimos años, entre ellos se pueden mencionar aplicaciones dedicadas a:

- Educación (para lo que fue creada la RPi)
- Domótica y automatización: desarrollo de un sistema flexible y de bajo costo para la motorización y control del hogar [23].
- Seguridad: sistema de monitorización y seguimiento para la vigilancia de sitios públicos [24].
- Streaming de Audio/Video: en conjunto con cámaras de seguridad para realizar streaming de vídeo y evitar el uso de computadoras o laptop para tal fin [25].
- Agricultura: tecnologías IoT para el armado de granjas inteligentes [26]; sistema de monitoreo del medio ambiente [27]
- Ingeniería: sistema de detección automática y notificación de baches en las calles [28].
- Seguridad Informática: desarrollo de un sistema de detección de intrusos utilizando RPi Honeypot [29].

- Salud: procesamiento de imágenes para mejorar el diagnostico de retinopatía diabética [30]; sistema para monitorización de signos vitales[31].

Estos proyectos muestran la versatilidad que la RPi tiene para ser aplicada en diferentes campos.

3.4. Familias y modelos

Las RPi tienen 2 familias: (1) la RPi original y (2) la RPi Zero. Esta segunda familia se puede ver como una versión reducida de la anterior, de menor costo, potencia, funcionalidad y tamaño.

3.4.1. Familia Raspberry Pi

Como se mencionó en la sección 3.2 existe en el mercado una amplia gama de variantes de las placas RPi. La familia RPi tiene 3 versiones, siendo el modelo actual la RPi 3 modelo B. La primera RPi, versión 1, tuvo varios modelos: A, B y B+, siendo B el más habitual. Las versiones 2 y 3 sólo tienen modelo B.

A continuación se enumera cada una de ellas con sus principales características.

Raspberry Pi

Este modelo esta compuesto por un SoC Broadcom BCM2835, chip gráfico VideoCore IV y procesador ARM1176JZF-S de un núcleo a 700MHz, que con técnicas de overclocking puede elevar la frecuencia hasta 1000 MHz.

- El modelo A cuenta con 256Mb de SDRAM, 1 puerto USB, y carece de conectividad Ethernet.
- El modelo B cuenta con 512Mb, 2 puertos USB, y añade un puerto Ethernet.
- Por su parte, el modelo B+ amplia a 4 los puertos USB, y cambia la tarjeta SD por una micro SD.
- Todos estos modelos contienen salidas de video RCA, HDMI, y DSI para un panel LCD.
- Como salidas de audio tienen un conector Jack de 3.5mm y salida por el puerto HDMI.

- Como dispositivos electrónicos, todos los modelos disponen de 8 x GPIO, SPI, I^2C y UART.

Figura 3.1: Raspberry Pi Modelo A

Raspberry Pi 2

La RPi 2 B fue una versión actualizada del Raspberry Pi B, cuyo principal cambio fue una potencia de cálculo superior. Compuesto por un SoC Broadcom BCM2836, un procesador ARM Cortex A7 de cuatro núcleos a 900 MHz y 1Gb de SDRAM (compartido con la gráfica).

- Se mantuvo el chip gráfico VideoCore IV.
- Cuenta con 4 puertos USB, puerto Ethernet 10/100 Mb.
- La tarjeta de memoria es Micro SD.
- El número de pines GPIO se amplía a 17, manteniendo las funciones SPI, I^2C y UART.
- Se eliminó la conexión RCA.

Raspberry Pi 3

El modelo RPi 3 B buscó fundamentalmente un cambio de conectividad, ya que la principal novedad fue la inclusión de Bluetooth 4.1 y Wifi 802.11n. La potencia se amplio con un SoC Broadcom BCM2837, y un procesador ARMv8 de cuatro núcleos a 1.2GHz de 64 bits.

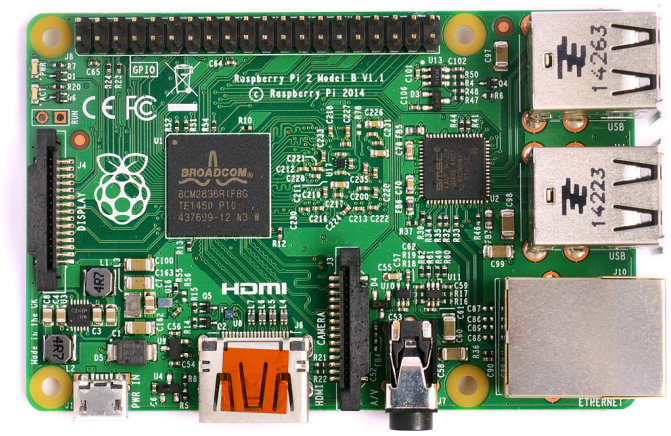


Figura 3.2: Raspberry Pi 2 Modelo B

- Se mantuvo el chip gráfico VideoCore IV
- 1Gb de SDRAM.
- Puerto Ethernet 10/100 Mb.
- 4 puertos USB.
- 17 GPIO con funciones SPI, I^2C y UART.

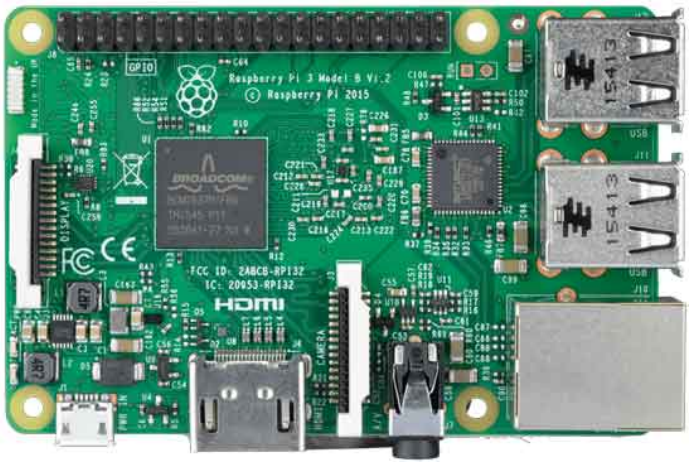


Figura 3.3: Raspberry Pi 3 Modelo B

3.4.2. Familia Raspberry Pi Zero

Esta familia se compone de una serie de modelos mas pequeños y de menor costo, que los hacen interesantes para integración en dispositivos IoT.

Raspberry Pi Zero

La RPi Zero tiene, a grandes rasgos, la misma potencia que una RPi 1 B, en un tamaño muy inferior. Al igual que la RPi 1 B, la RPi Zero incluye el SoC Broadcom BCM2835, procesador ARM1176JZF-S a 1Ghz, y cuenta con 512Mb de SDRAM.

Dado el pequeño tamaño, prescinde del puerto de Ethernet y el conector DSI, y cuenta únicamente con un puerto Micro USB. Mantiene, sin embargo, las funciones electrónicas GPIO, SPI, I2C y UART.

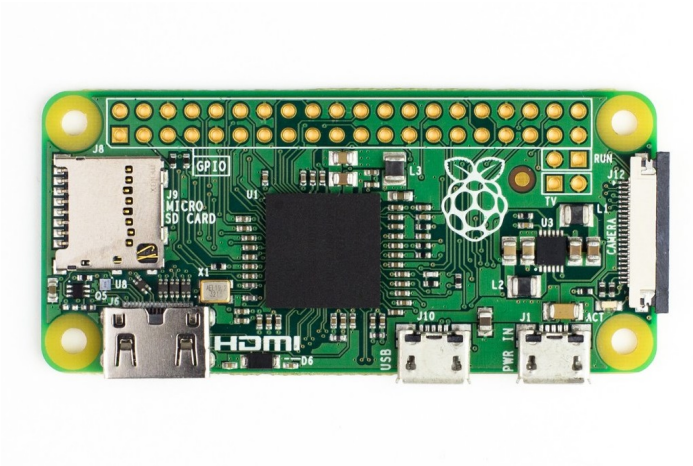


Figura 3.4: Raspberry Pi Zero

Raspberry Pi Zero W

Se puede considerar una actualización de la RPi Zero original que añade Bluetooth 4.1 y Wifi 802.11n, manteniendo el resto de características.

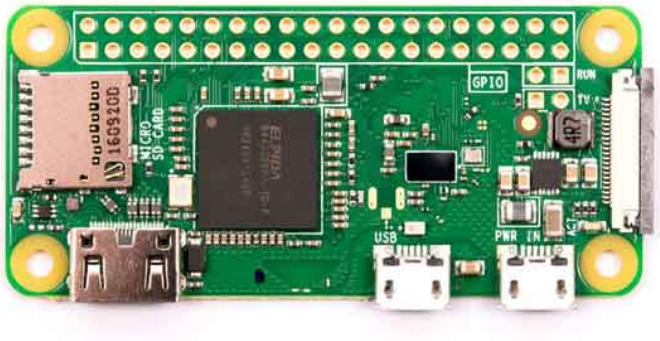


Figura 3.5: Raspberry Pi Zero W

3.4.3. Resumen comparativo

A continuación en la Tabla 3.1 se muestra las características de los diferentes modelos en forma comparativa.

3.5. Raspberry Pi 3

En esta sección se describe la arquitectura de la RPi 3 modelo B, dado que fue la elegida para el despliegue del cluster correspondiente a esta tesina.

La RPi 3 está conformada por un SoC especialmente desarrollado para este modelo: el Broadcom BCM2837. Este SoC incluye 4 núcleos de procesamiento ARM Cortex-A53 de alto rendimiento que funcionan a 1.2GHz con 32Kb de cache de nivel 1 y 512Kb de nivel 2, sumado a un procesador gráfico VideoCore IV que se encuentra conectado a un módulo de memoria LPDDR2 de 1 GB en la parte posterior de la placa.

Este modelo comparte el mismo chip SMSC LAN9514 que su predecesor, la RPi 2, agregando conectividad Ethernet 10/100 y cuatro canales USB a la placa. El chip SMSC se conecta al SoC a través de un único canal USB, que funciona como un adaptador USB-a-Ethernet y un concentrador USB.

También se puede encontrar el chip Broadcom BCM43438 que proporciona conectividad LAN inalámbrica de 2.4GHz 802.11n, y Bluetooth 4.1 de bajo consumo. Es importante destacar que no es necesario conectar una antena externa a la RPi 3, dado que las señales inalámbricas son emitidas y recibidas por una antena tipo chip soldada directamente a la placa, a fin de mantener el tamaño del dispositivo al mínimo. A pesar de su tamaño, esta antena presenta excelente rendimiento.

3.6. Sistemas operativos y software disponible

Con los primeros modelos y versiones de la RPi, la experiencia de usuario como sistema de escritorio resultaba problemática. Con las mejoras realizas en el hardware y en los complementos sumado a los sistemas disponibles para las placas, cada vez más personas eligen utilizar las RPi como PC.

A continuación se mencionaran los sistemas operativos más elegidos por los usuarios a la hora de utilizar estas placas.

Modelo	RPi B+	RPi 2 B	RPi 3 B	RPi Zero	RPi Zero W
Fecha de lanzamiento	Feb 2012	Feb 2015	Feb 2016	Nov 2015	Feb 2017
Precio (US$)	25	35	35	5	10
SoC	Broadcom BCM2835	Broadcom BCM2836	Broadcom BCM2837	Broadcom BCM2835	Broadcom BCM2835
Procesador	ARM1176JZF-S	Cortex-A7	Cortex-A53 64-bit	ARM1176JZF-S	ARM1176JZF-S
Nº cores	1	4	4	1	1
GPU	VideoCore IV	VideoCore IV	VideoCore IV	VideoCore IV	VideoCore IV
CPU Clock	700 MHz	900 MHz	1.2 GHz	1 GHz	1 GHz
RAM	512 MB @ 400 MHz	1 GB @ 400 MHz	1 GB @ 400 MHz	512 MB @ 400 MHz	512 MB @ 400 MHz
Memoria	Micro SD	Micro SD	Micro SD	Micro SD	Micro SD
USB 2.0	2	4	4	1 microUSB	1 microUSB
Ethernet	Sí (10/100)	Sí (10/100)	Sí (10/100)	No	No
Wi-Fi	No	No	Si	No	Si
Bluetooth	No	No	Si	No	Si
HDMI	Si	Si	Si	Mini	Mini
GPIO	8	17	17	17	17
UART	Si	Si	Si	Si	Si
SPI	Si	Si	Si	Si	Si
I2C	Si	Si	Si	Si	Si
DSI (LCD)	Si	Si	Si	No	No
Cámara	Si	Si	Si	Si	Si
Altura	85.6 mm	85.6 mm	85.6 mm	65 mm	65 mm
Ancho	53.98 mm	56.5 mm	56.5 mm	30 mm	30 mm
Profundidad	17 mm	17 mm	17 mm	5 mm	5 mm
Peso	45 g	45 g	45 g	9 g	9 g
Consumo[a]	330 mA @ 5V	350 mA @ 5V	400 mA @ 5V	100 mA @ 5V	150 mA @ 5V

Tabla 3.1: Tabla comparativa de las características principales de los modelos de Raspberry Pi.

[a]El valor expresado corresponde al de la placa sin dispositivos USB conectados.

Raspbian OS

Raspbian OS es un sistema operativo gratuito basado en Debian el cual está optimizado para RPi. Raspbian proporciona más que un sistema operativo puro: viene con más de 35.000 paquetes y diversos softwares precompilados de fácil instalación en la RPi. Sin embargo, Raspbian aún se encuentra en desarrollo activo con énfasis en mejorar la estabilidad y el rendimiento del mayor número posible de paquetes Debian.

Si bien es el sistema operativo recomendado por la Fundacion Raspberry Pi, Raspbian no está afiliado a la misma. Raspbian fue creado por un equipo pequeño y dedicado de desarrolladores que son entusiastas del hardware RPi, de los objetivos educativos de la Fundación Raspberry Pi y del Proyecto Debian.

En la Figura 3.6 puede observarse una captura de pantalla del escritorio de Raspbian con algunas de las aplicaciones que lo integran.

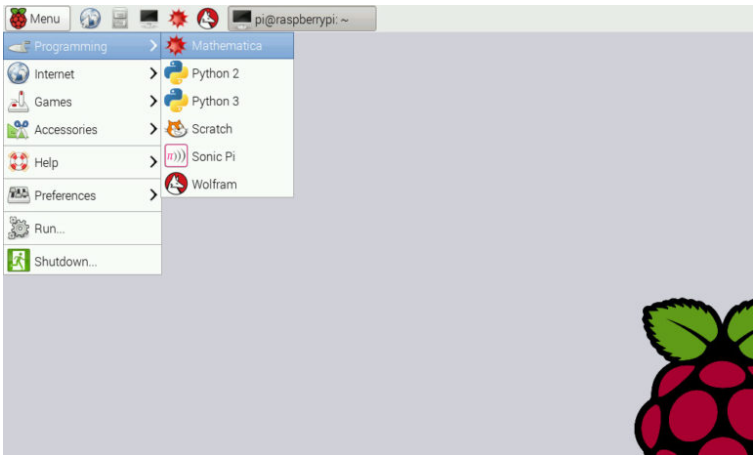


Figura 3.6: Entorno de escritorio PIXEL en Raspbian.

Ubuntu MATE

La famosa distribución Ubuntu de Canonical, también tiene disponible Ubuntu MATE para la RPi. Se trata de un Ubuntu Linux con un entorno de escritorio MATE ligero en comparación con el pesado escritorio Unity. MATE se basa en GNOME2 y es muy conocido en el mundo Linux por su gran aceptación dentro de la comunidad, tanto es así, que existen numerosas distribuciones que lo han elegido como escritorio por defecto.

Con Ubuntu MATE, se dispone de una completa distribución preparada para realizar una amplia gama de tareas. Hay que tener en cuenta que al ser una distribución

potente, exige que al menos se instale en una RPi 2 o superior. En la Figura 3.7 se observa la pantalla principal de Mate.

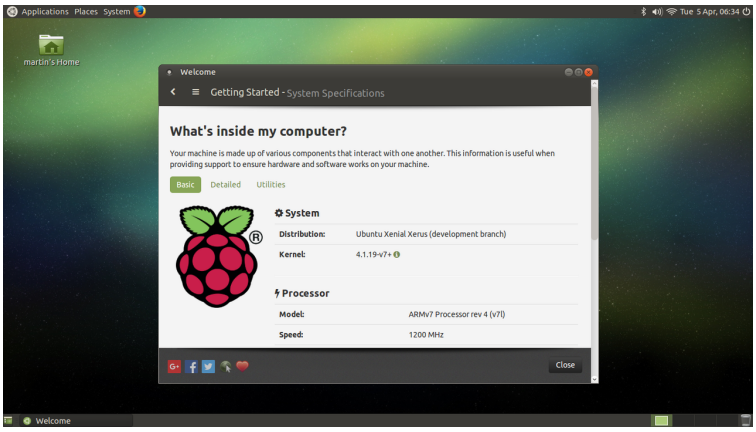


Figura 3.7: Ubuntu Mate

Windows 10 IoT Core

Windows 10 IoT Core es una versión reducida de Windows 10 para la plataforma ARM. Muchas de sus características conocidas no se incluyen porque no está pensada para usuarios finales sino para desarrolladores y *makers* a los cuales le permite construir dispositivos de bajo costo con menos recursos.

A diferencia de las versiones Linux, carece de una interfaz de usuario del sistema tipo escritorio. A pesar de esto, IoT Core continúa siendo un sistema operativo completamente funcional que, en conjunto con Visual Studio y las APIs centrales de Windows, logran que una RPi aproveche la plataforma universal Windows.

Es importante destacar que este sistema operativo es gratuito, totalmente compatible con Microsoft y recibe actualizaciones periódicas. En la Figura 3.8 se aprecia las limitadas opciones de la interfaz que tiene este sistema operativo.

3.7. Comparación teórica con procesadores x86

El consumo de energía en los sistemas embebidos es uno de los criterios de diseño más importantes. Un sistema que está diseñado para conectarse a una fuente de alimentación, como la red eléctrica, normalmente puede ignorar las limitaciones del consumo de energía, pero un diseño móvil (o uno conectado a una fuente de alimentación poco fiable) puede depender totalmente de la gestión de la energía.

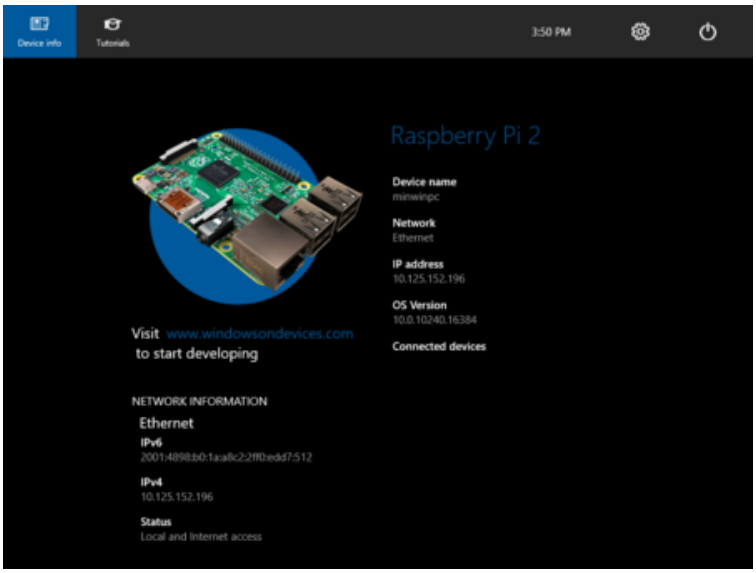


Figura 3.8: W10 IoT Core

Los procesadores ARM sobresalen por sus diseños de bajo requerimiento de potencia, no requiriendo de disipadores térmicos. Su consumo de potencia es inferior a 5W, aun cuando incluyen GPU, periféricos y memorias.

Esta pequeña disipación de potencia solo es posible gracias a una menor cantidad de transistores utilizados y a las frecuencias de reloj relativamente más bajas (comparadas con las CPUs de escritorio comunes). Esta característica, repercute en el rendimiento del sistema y, por lo tanto, ciertas operaciones pueden requerir mas tiempo.

La Tabla 3.2 presenta algunas características relacionadas con el rendimiento y el consumo de potencia de algunos modelos que pueden ser considerados representativos de las diferentes arquitecturas mencionadas a lo largo de este trabajo. Para medir el rendimiento se ha usado la métrica FLOPS (precisión simple) mientras que para la eficiencia energética se ha empleado la métrica FLOPS/Watt. Como no existe un método universal para realizar comparaciones entre arquitecturas, la misma puede ser injusta en algunos aspectos. Aun así, resulta útil para mostrar características distintivas y valores de referencia de cada una.

Es importante aclarar que al utilizar el pico de rendimiento para realizar los cálculos de eficiencia energética, solo es posible realizar una comparación teórica entre unidades de procesamiento. El rendimiento y la eficiencia energética que una plataforma puede alcanzar depende fuertemente del problema a resolver y de la capacidad que tenga el software de aprovechar sus características.

Nombre	Núcleos	Frec.(GHz)	FLOPS/ciclo	GFLOPS	Watt	GFLOPS/W
RPi 2	4	0.9	2	7.2	1,75	4,11
RPi 3	4	1.2	8	38.4	2	19,2
RPi Zero (ARM11)	1	1	4	4	0,5	8
Intel Core i7-7700	4	3.7	32	236.8	65	3.64
Intel Xeon Gold 6146	12	2.7	32	1036.8	165	6.28
AMD Ryzen 3 2200G	8	3.5	8	224	65	3.44
AMD EPYC 7351P	16	2.9	16	792.4	170	4.36

Tabla 3.2: Comparativa entre procesadores según su rendimiento energético.

Como se puede observar en la Tabla 3.2, los procesadores x86 son capaces de obtener picos de rendimiento muy superiores a los de la RPi 3, especialmente aquellos que pertenecen al segmento de alto rendimiento como el Xeon Gold 6146 y el EPYC 7351P. Aunque el pico de rendimiento de la RPi 3 es bastante más bajo que el del resto, su bajo consumo de potencia, las vuelve la opción más eficiente desde el punto de vista energético.

3.8. Resumen

La Raspberry Pi (RPi) es un SBC que tiene el tamaño de una tarjeta de crédito, desarrollada por la fundación Raspberry Pi en Reino Unido. La idea de la computadora RPi surge en el año 2006. En esa época, Eben Upton trabajaba en el Cambridge Computer Laboratory de Inglaterra. Junto a varios de sus colegas del laboratorio, Eben notó que, en los últimos años, los ingresantes a la carrera de Ciencias de la Computación tenían pocos conocimientos del funcionamiento interno de una computadora. Surge entonces la idea de recrear el ecosistema de las microcomputadoras de los 80s que permitían una comprensión total del hardware/software y fomentaban la programación y modificación de la computadora, en vez del simple uso del software.

Gracias a la versatilidad de estas placas puede ser utilizada en distintos ámbitos: educación (para lo que fue creada), medicina, seguridad, domótica, robótica, agricultura, ingeniería, seguridad informática, entre otras.

Existe en el mercado una amplia gama de variantes a las placas RPi. La familia RPi tiene 3 versiones, siendo el modelo actual la RPi 3 modelo B. La primera RPi, versión 1, tuvo varios modelos, A, B y B+, siendo B el más habitual. Las versiones 2

y 3 sólo tienen modelo B.

Para esta tesina se utilizó la versión 3 modelo B que cuenta con un SOC Broadcom BCM2837, y un procesador ARMv8 de cuatro núcleos a 1.2GHz de 64 bits, además de un el chip Broadcom BCM43438 que proporciona conectividad LAN inalámbrica de 2.4GHz 802.11n, y Bluetooth 4.1 de bajo consumo.

En cuanto a los sistemas operativos adaptados a estas placas se tiene como alternativa a Raspbian OS, Ubuntu Mate y Windows 10 IoT Core; siendo Raspbian la distribución más elegida por la comunidad.

Aunque el pico de rendimiento de la RPi 3 es bastante más bajo que el de los procesadores x86, su bajo consumo de potencia las vuelve la opción más eficiente desde el punto de vista energético.

Capítulo 4

Despliegue del cluster

Ya hemos visto en el Capítulo 2 los beneficios que brinda la computación paralela. En ocasiones, construir un cluster para realizar pruebas requiere el uso de hardware comercial costoso y de gran tamaño, como computadoras de escritorio, servidores, o la implementación de configuraciones complejas en maquinas virtuales. Gracias al bajo costo y al reducido tamaño de las placas RPi, construir un cluster para explorar el mundo de la computación paralela hace que esto sea accesible y fácil de llevar a cabo para un usuario con pocos conocimientos sobre el tema.

Si bien las RPi no son la opción adecuada para un sistema complejo de producción, resulta interesante el conjunto de herramientas que brinda para aprender las tecnologías que conforman los cluster profesionales. Por ejemplo, permite trabajar con estándares de la industria, como MPI u otros proyectos de vanguardia de código abierto.

En este Capítulo se explican algunos conceptos de la implementación de computación paralela sobre un cluster y el conjunto de tecnologías asociada a ella, además de proporcionar una introducción al uso de RPi. Por último, se describe paso a paso el proceso llevado a cabo para el despliegue del cluster empleado en esta tesina. En la Figura 4.1 se muestra la imagen del cluster de RPi desplegado.

4.1. Descripción del hardware

Un cluster está formado por un conjunto de dispositivos conectados entre sí para que se puedan comunicar. Para realizar el despliegue del mismo se requieren una serie de elementos: los dispositivos encargados de realizar el cómputo y el control (nodos); un dispositivo que brinde un medio de comunicación que permita interconectar los

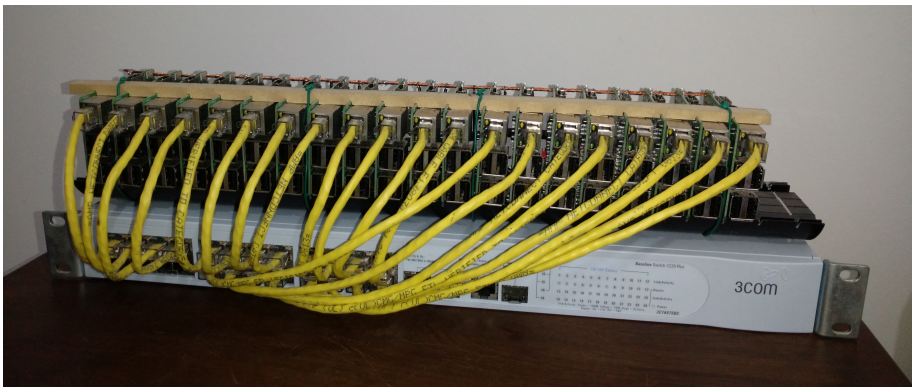


Figura 4.1: Cluster de RPi desplegado

mismos; y una fuente de poder que permita satisfacer la demanda energética de los dispositivos de cómputo. Para que los dispositivos se comuniquen entre sí se debe configurar el hardware de red y realizar las conexiones de las placas al mismo por medio de cables. También se necesita contar con hardware complementario el cual facilita la tarea de configuración de los dispositivos (una vez finalizada esta etapa se podrá prescindir del mismo).

A continuación se proporciona un listado de los elementos que se utilizaron para realizar el despliegue del cluster:

- Veinte RPi 3 Modelo B (las características de estas placas se trataron en el Capítulo 3).
- Una TV o Monitor con interfaz VGA/HDMI con el apropiado adaptador o cable.
- Un teclado USB.
- Una fuente de PC de al menos 250W.
- Veinte cables Ethernet/RJ45.
- Un Switch 3COM 3C16475BS Baseline 2226 Plus.
- Veinte tarjetas Micro SD Verbatim de 16Gb.
- Una Laptop o PC de escritorio para realizar las conexiones remotas.
- Una conexión de Internet existente (preferentemente del tipo WiFi).

4.1.1. TV o Monitor VGA/HDMI

La RPi 3 está provista de una interfaz HDMI, que permite conectarla a una TV HD o a Monitores HD. Es recomendable conectarla a uno de estos dispositivos utilizando un cable HDMI y evitar el uso de adaptadores HDMI/VGA dado que los mismos pueden presentar incompatibilidad con la frecuencia de refresco que utilizan las RPi. Cabe mencionar, que hasta el día de la fecha existe un bug en las placas que hace que las mismas no emitan señal de vídeo si se conecta el cable HDMI posteriormente al encendido de la placa. Para evitar esto se debe conectar el cable HDMI antes de encender la placa.

4.1.2. Teclado USB

Se debe utilizar un teclado USB estándar, para realizar las configuraciones iniciales de las placas. Una vez que se configura el servicio SSH en el sistema operativo para aceptar conexiones de acceso remoto, se puede prescindir de este dispositivo.

4.1.3. Fuente de PC de 250W

Al momento de brindar alimentación a las placas, se puede optar por dos alternativas:

- Alimentación individual
- Alimentación global

En la primera opción, es necesario contar con tantas fuentes de alimentación como nodos vayan a componer el cluster. La principal desventaja es la necesitad de disponer de las fuentes de alimentación (20 en el caso del cluster de esta tesina) y un medio donde conectarlas a la red eléctrica (por ejemplo zapatillas). Una ventaja es la de poder desconectar las fuentes de alimentación de aquellas placas que no se estén utilizando. Otra ventaja es la simpleza de realizar las conexiones: sólo se debe conectar la fuente de alimentación en el conector USB Micro-B de la placa RPi.

En el caso de la alimentación global, se busca alimentar todas las placas RPi al mismo tiempo. Para esto se debe contar con una fuente de alimentación capaz de soportar la potencia total consumida por las placas. Es importante tener en cuenta dos características. La primera es para alimentar una placa RPi se necesita una fuente de alimentación que sea capaz de brindar 5.1V a 2.5A. La segunda es que debido a

que hay que proveer de energía a 20 placas, se necesita contar con una fuente de alimentación de al menos 50A (20 placas × 2.5A c/u) en su salida.

Para la realización del cluster, se optó por la alimentación global, alimentando las placas RPi con una fuente estándar de PC de 250W, al contar con un gran número de placas para alimentar, resulta más práctica que la alimentación individual. En la Figura 4.2 se puede ver el detalle del medio común de alimentación de las placas.

Figura 4.2: Alimentación del cluster en detalle.

4.1.4. Switch 3COM Baseline 2226 Plus (3C16475BS)

Un switch es un dispositivo que vincula múltiples máquinas en una misma red. Al trabajar con placas RPi, resulta suficiente usar un switch que permita manejar velocidades de 10/100 Mbps teniendo en cuenta su limitación de velocidad de red. En el despliegue realizado en esta tesina se utilizó un Switch 3COM Baseline 2226 Plus (3C16475BS) de 24 bocas.

4.1.5. Cables Ethernet/RJ45

Cables empleados para conectar las placas RPi al Switch.

4.1.6. Micro SD Verbatim de 16Gb

La RPi requiere de una tarjeta MicroSD para iniciar su sistema. Además, la misma también será usada como dispositivo de almacenamiento de los distintos archivos que generen las pruebas realizadas.

4.1.7. Laptop o PC de escritorio

Se necesitará contar con una computadora conectada al mismo switch del cluster. Al conectar una computadora al mismo switch del cluster, no se necesita que las placas estén conectadas a un monitor/TV y un teclado para ejecutar comandos en la terminal. Para hacer esto, vía SSH (Secure SHell) [32] se realizarán las conexiones a las diferentes placas.

4.1.8. Conexión a Internet

El enlace de Internet permite descargar los paquetes de las herramientas que se utilizan para configurar los servicios del cluster.

4.2. Configuración general

En esta sección se detalla como instalar el sistema operativo en la RPi y como configurar los parámetros básicos para los nodos del cluster.

4.2.1. Instalación de Raspbian

Se utilizó Raspbian en su versión Lite como sistema operativo en las placas RPi. Lo primero es, descargar de la página oficial de Raspberry [5] la imagen correspondiente a la versión de Raspbian Stretch Lite. Se eligió esta distribución porque es práctica, liviana, es recomendada por los creadores de las placas RPi, tiene comunidad de usuarios amplia y, principalmente, porque cuenta solamente con los servicios básicos del sistema. De esta forma, se tiene la mayor parte de los recursos del sistema disponibles y permite hacer el SO totalmente personalizable.

Una vez finalizada la descarga, se obtiene un archivo con extensión *zip* del cual se descomprime un archivo con extensión *img*. Este archivo contiene una imagen de disco el cual debe ser grabada dentro de la MicroSD. En la Figura 4.3 se observa las versiones disponibles de Raspbian en el sitio web de Raspberry.

Para escribir el archivo *img* dentro de la MicroSD se utilizará el software Etcher, que es recomendado en el sitio oficial de Raspberry. Etcher es una aplicación gratuita y de código abierto desarrollada para facilitar la grabación de imágenes ISO, IMG y otros formatos comprimidos directamente y de forma segura a memorias USB y tarjetas de memoria. Las principales características de esta aplicación son:

- ofrece una interfaz sencilla y agradable.

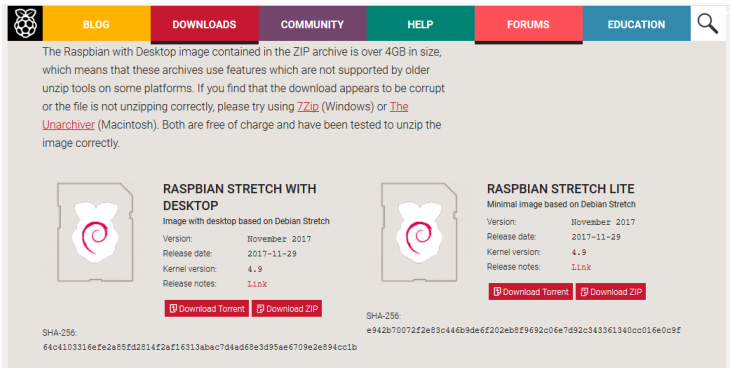


Figura 4.3: Versiones disponibles de Raspbian.

- cuenta con sistemas de validación para evitar la creación de memorias corruptas.
- facilita la elección de la unidad USB de destino evitando equivocarnos y borrar por error un disco duro.
- escrito completamente en JS, HTML, Node.js y Electron.
- es multi-plataforma, se puede descargar para Windows, Mac OS X y Linux.

Como se observa en la Figura 4.4, el proceso de copiado consta de 3 pasos:

1. *Select Image* – Seleccionar la imagen ISO, IMG o cualquier otro formato compatible.
2. *Select Drive* – Elegir la unidad de destino donde se grabara la imagen seleccionada en el paso 1.
3. *Flash!* – Si todo está correcto, se inicia la grabación.

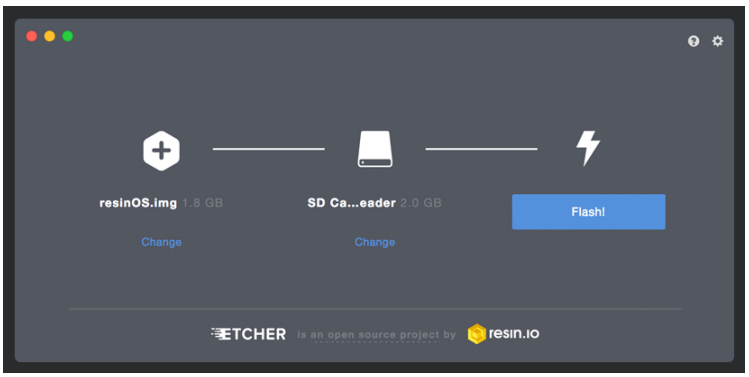


Figura 4.4: Etcher y su simple e intuitiva interfaz.

El proceso de grabación de la imagen puede tardar varios minutos, esto depende del tamaño del archivo img y la velocidad de la tarjeta. Una vez que finalice la copia de los datos, Etcher verifica que todo está correcto y no se ha generado un medio corrupto. Finalizada la grabación, la tarjeta MicroSD se encuentra en condiciones de ser insertada en la RPi.

4.2.2. Encendido de la Raspberry Pi

Basta conectar la Raspberry Pi a la alimentación para que la misma se encienda. Previamente se debe haber conectado el teclado y el dispositivo de vídeo elegido (TV o monitor).

A diferencia de las versiones anteriores de Raspbian, la expansión del sistema de archivos es automática y la misma es realizada durante el primer arranque de la RPi. La expansión del sistema de archivos se realiza porque al momento de grabar la imagen de Raspbian en la tarjeta MicroSD, esta tiene el tamaño mínimo (1.7Gb aproximadamente). La expansión se lleva a cabo para poder aprovechar el tamaño máximo de la tarjeta (16Gb en nuestro caso). Una vez realizada la expansión, la RPi se reinicia y, finalizado el proceso de arranque, estaremos en condiciones de hacer las primeras configuraciones.

4.2.3. Configuración inicial

Antes de empezar a configurar los distintos tipos de nodos (master y workers), existen un conjunto de configuraciones comunes que se deben realizar. Lo primero que se necesita hacer en la Raspberry es autenticarse; para eso se deben ingresar las credenciales de acceso provistas por el fabricante que son usuario: *pi,* y la contraseña: *raspberry.*

Una vez que se tenga el control de la terminal ya se podrá ingresar los comandos para configurar la Raspberry. A continuación, se deben elevar los privilegios del usuario *pi,* lo cual es necesario debido a que este usuario no cuenta con los permisos requeridos para realizar modificaciones en la configuración del sistema operativo. Para eso se ejecuta el comando `sudo su`, logrando obtener los privilegios de usuario *root* o superusuario.

El proceso de configuración se inicia ingresando los parámetros para establecer una conexión WIFI. Para esto se deben ejecutar en la consola los siguientes comandos:

```
cd /etc/wpa_supplicant
```

```
wpa_passphrase "nombre_SSID" "contraseña_SSID" >> wpa_supplicant.conf
```

A continuación se analiza el comando en detalle:

`wpa_passphrase` genera la salida que debe contener el archivo wpa_supplicant.conf

`nombre_SSID` Nombre de la red WIFI a la que se conectará la placa

`contraseña_SSID` Contraseña de la red WIFI a la que se conectará la placa

Este comando permite configurar los parámetros de la red WiFi, la cual se utiliza para brindar conexión de Internet a la placa RPi. Para verificar la correcta configuración del archivo, se debe ejecutar el comando `cat /etc/wpa_supplicant/wpa_supplicant.conf`. Como se puede apreciar en la Figura 4.5 el comando `wpa_passphrase` genera las líneas de configuración de la red especificada, almacenando la contraseña cifrada, y una línea comentada con la contraseña en texto plano, que no tendrá efectos sobre la configuración.

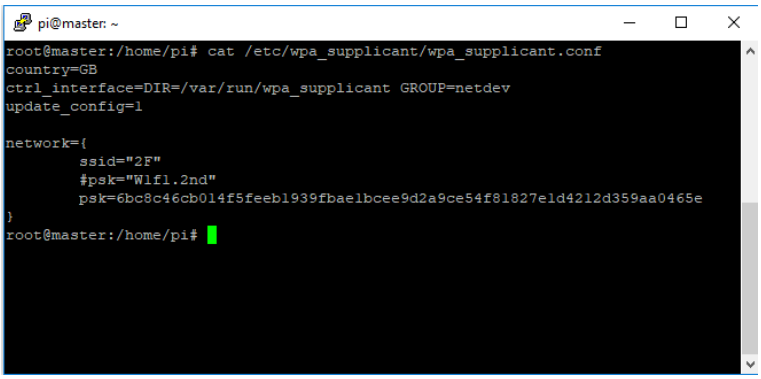


Figura 4.5: Contenido del archivo wpa_supplicant.conf

Se puede verificar si se ha conectado correctamente utilizando el comando `ifconfig wlan0`. Si en la salida de este comando el campo `inet addr` tiene una dirección IP al lado, la Raspberry Pi se ha conectado a la red. De lo contrario, debemos verificar que la contraseña y el SSID ingresados en el archivo wpa_supplicant sean correctos.

Al contar con una conexión a Internet, el siguiente paso será ejecutar el comando `raspi-config`. Con este comando se visualiza un menú interactivo (ver Figura 4.6) que simplifica la tarea de hacer modificaciones del sistema, sin tener que editar archivos a mano como se hizo anteriormente con el archivo wpa_supplicant.conf.

Lo primero que se debe realizar es la actualización de los paquetes del sistema. Para esto se debe seleccionar la opción *8 Update* del menú mencionado (ver Figura 4.5), lo que inicia la actualización de los paquetes del sistema.

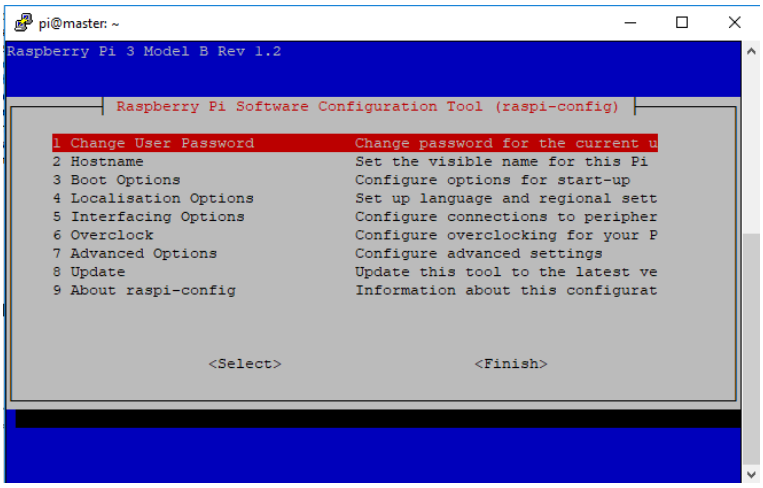


Figura 4.6: Menú del comando *raspi-config*.

Finalizada la actualización, se volverá a visualizar el menú. El siguiente paso será configurar el acceso remoto y maximizar el tamaño de la memoria RAM, para llevar esto a cabo se deben seguir los siguiente pasos:

Acceso remoto: seleccionar la opción *5 Interfacing Opctions*. En la siguiente pantalla, seleccionar la opción *P2 SSH* y por último confirmar la operación seleccionando la opción *Yes*. A continuación se visualiza un mensaje confirmando la habilitación del servicio SSH, el cual permitirá iniciar conexiones remotas a la placa Raspberry Pi.

Maximizar la memoria RAM: configurar el tamaño de la memoria asignada a la GPU, lo que permite tener la mayor cantidad de memoria RAM para el momento de realizar las pruebas. Para realizar esta configuración se debe seleccionar la opción *7 Advanced Options*, y a continuación seleccionar la opción *A3 Memory Split*. En la siguiente ventana se solicita ingresar el nuevo tamaño de la memoria el cual será *16*, que es el mínimo de memoria asignable para gráficos. Aceptar para confirmar la operación.

Para garantizar que las configuraciones realizadas surtan efecto, se debe finalizar el asistente de configuración. Esto se consigue al seleccionar la opción *Finish* y confirmando el reinicio del sistema.

Con este último paso se completa la configuración básica de la primera RPi, la cual servirá como punto de partida para configurar los nodos del cluster. Para evitar repetir este proceso para cada placa restante, es recomendable generar una imagen del sistema almacenado en tarjeta MicroSD.

4.3. Configuración del nodo master

Como se mencionó anteriormente, el nodo master es el equipo desde el que se administra el cluster. En él se llevan a cabo tareas como lanzamiento de trabajos, instalación de librerías y programas, obtención de los resultados de las ejecución de las aplicaciones, etc. Además, es el punto de acceso al resto de los equipos que conforman el cluster.

A continuación se realiza una breve descripción de los elementos mínimos con los que deberá contar el nodo master para administrar el cluster (los mismos serán desarrollados posteriormente en detalle en esta sección):

Generación de claves para permitir la autenticación por clave pública y evitar la autenticación con usuario y contraseña.

Configuración de red necesaria para permitir la interconexión con el resto de los nodos.

Protocolo de configuración dinámica de host (DHCP, por sus siglas en inglés) necesario para el aprovisionamiento de los parámetros de red a cada uno de los nodos workers.

Sistema de archivos de red (NFS, por sus siglas en inglés) permite compartir cualquier archivo o programa que se desee a través de la red, para dejarlo accesible desde todos los nodos del cluster.

Traducción de direcciones de red (NAT, de sus siglas en inglés) permite utilizar el nodo master como un punto de acceso a Internet para cada uno de los nodos.

Librerías MPI necesarias para compilar los algoritmos paralelos que serán ejecutados en el cluster.

A continuación se inicia la configuración del nodo master partiendo de la configuración básica descrita en la sección 4.2.3.

4.3.1. Generación de claves

Desde una PC conectada a la misma red WIFI configurada anteriormente, conectarse a la RPi utilizando la IP de la misma por medio de un cliente SSH. Notar que el sistema operativo solicita las credenciales de acceso, utilizar las mismas que se utilizaron para hacer el primer ingreso al sistema (sección 4.2.3).

SSH permite utilizar una amplia variedad de mecanismos de autenticación diferentes entre los que se incluyen usuario y contraseña. Posiblemente este sea el mecanismo más utilizado por muchos para autenticar una sesión SSH, pero SSH también soporta la autenticación mediante clave pública. La autenticación con clave pública funciona con dos claves: una pública y otra privada. Para entender el funcionamiento imaginemos que se cuenta con un candado y su llave. La clave pública funciona como un candado y la privada como la llave. El candado se colocará en el dispositivo remoto al que se quiere acceder; cuando se intenta acceder se comprobará que la máquina que intenta conectar tiene la llave, que en este caso resulta ser la clave privada.

Para configurar el acceso SSH con clave pública se debe:

- Generar el par de claves pública/privada.
- Copiar la clave pública a cada uno de los nodos.

Para generar las claves ejecutaremos el siguiente comando en la terminal:

```
ssh-keygen -t rsa
```

Como resultado de la ejecución del comando `ssh-keygen` se generan dos archivos:

id_rsa es la clave privada, la que permanecerá en la máquina local.

id_rsa.pub es la clave pública, la que se tiene que copiar a cada nodo worker.

Durante la generación de claves, el sistema, solicita asignar un *passphrase* o contraseña para proteger la clave privada. Sin embargo, como OpenMPI requiere un *passphrase* vacío, no se asignará ninguno.

Una vez generado el par de claves, hay que copiar la clave pública en cada nodo remoto (en la sección 4.4 se explican los pasos a seguir).

4.3.2. Configuración de red

Para que todos los nodos puedan funcionar como un cluster, deben estar interconectados entre sí. La Figura 4.7 ilustra la arquitectura de red en la cual se encuentra basada el cluster. En la imagen, podemos identificar al nodo *master*, con sus servicios y los 19 nodos *workers*.

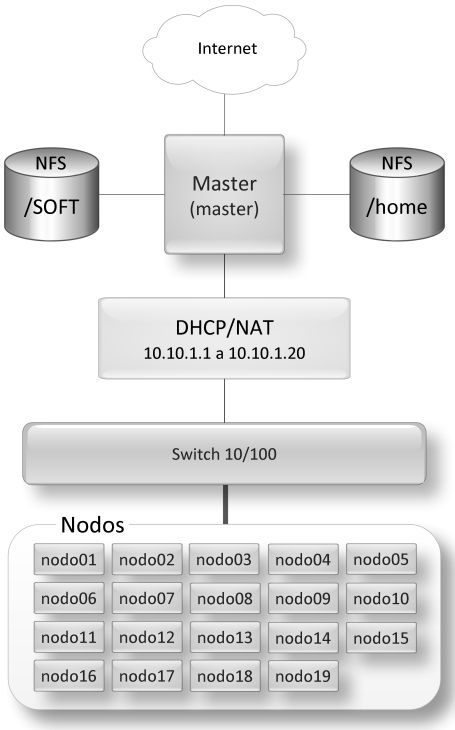


Figura 4.7: Arquitectura de red del cluster Raspberry Pi.

El primer paso es modificar el nombre de equipo del sistema. La función de este es, principalmente, la de ayudar a identificar el equipo dentro de una red.

Una vez adentro del sistema, se debe ejecutar el comando *sudo su* para obtener permisos de usuario *root*. Luego, ejecutar el comando *raspi-config* para visualizar el asistente de configuración de RPi (ver Figura 4.6), seleccionar la opción *2 Hostname;* e ingresar como nombre de equipo: *master*. se debe confirmar la opetación y finalizar el asistente.

El siguiente paso será modificar el archivo `/etc/hosts`. Este archivo se utiliza para obtener una relación entre un nombre de equipo y una dirección IP. En cada línea de `/etc/hosts` se especifica una dirección IP y los nombres de equipo que le corresponden, de forma de que no se tenga que recordar las direcciones de IP sino nombres de equipos. Habitualmente se suelen incluir las direcciones, nombres y alias de todos los equipos conectados a la red local, de manera de que para la comunicación dentro de la red no se tenga que recurrir a un DNS (Domain Name Server) a la hora de resolver un nombre de host. El formato de una línea de este fichero puede ser el siguiente:

```
10.10.1.254 master.cluster.rpi master
```

Esta línea indica que será equivalente utilizar la dirección `10.10.1.254`, el nombre de equipo `master`, o el alias `master.cluster.rpi` para hacer referencia a este equipo.

Para simplificar esta tarea se utiliza el Algoritmo 4.1 que permite generar en el archivo hosts todas las entradas correspondientes a los nodos del cluster.

Algoritmo 4.1: Bash Script para generar los nombres de equipos en el archivo /etc/hosts.

```
#!/bin/bash

#Se elimina el archivo hosts actual.
rm /etc/hosts

#Creamos las lineas correspondientes a los nodos del cluster.
cat << EOF >> /etc/hosts
10.10.1.254 master.cluster.rpi master
127.0.0.1 localhost
10.10.1.1 nodo01.cluster.rpi nodo01
10.10.1.2 nodo02.cluster.rpi nodo02
10.10.1.3 nodo03.cluster.rpi nodo03
10.10.1.4 nodo04.cluster.rpi nodo04
10.10.1.5 nodo05.cluster.rpi nodo05
10.10.1.6 nodo06.cluster.rpi nodo06
10.10.1.7 nodo07.cluster.rpi nodo07
10.10.1.8 nodo08.cluster.rpi nodo08
10.10.1.9 nodo09.cluster.rpi nodo09
10.10.1.10 nodo10.cluster.rpi nodo10
10.10.1.11 nodo11.cluster.rpi nodo11
10.10.1.12 nodo12.cluster.rpi nodo12
10.10.1.13 nodo13.cluster.rpi nodo13
10.10.1.14 nodo14.cluster.rpi nodo14
10.10.1.15 nodo15.cluster.rpi nodo15
10.10.1.16 nodo16.cluster.rpi nodo16
10.10.1.17 nodo17.cluster.rpi nodo17
10.10.1.18 nodo18.cluster.rpi nodo18
10.10.1.19 nodo19.cluster.rpi nodo19
EOF
```

Una vez generado el nuevo archivo hosts, el master será capaz de identificar los distintos nodos del cluster por medio de su nombre de equipo.

A diferencia de los nodos worker, el nodo master será el único nodo al que se le configure la intefaz Ethernet manualmente. Entonces, será necesario configurar la IP de la interfaz Ethernet. Para esto se debe editar el archivo /etc/dhcpcd.conf, y reemplar la linea 42 por: `static ip_address=10.10.1.254/24`.

4.3.3. DHCP

El siguiente paso será instalar un servidor DHCP. Este servicio permite que la configuración IP de los nodos se haga de forma automática, evitando así la necesidad de tener que realizar manualmente uno por uno la configuración de la interfaz de red de cada equipo.

El Algoritmo 4.2 instala y configura el servidor DHCP en el nodo master:

Algoritmo 4.2: Bash Script para configurar el servicio DHCP.

```
#!/bin/bash
#Instalamos el paquete del servidor.
apt install isc-dhcp-server -y

#Hacemos un backup del archivo de configuración original.
cp /etc/dhcp/dhcpd.conf{,.backup}

#Eliminamos el contenido del archivo de configuración.
cat /dev/null > /etc/dhcp/dhcpd.conf

#Editamos el archivo del servidor DHCP con nuestros parámetros.
cat << EOF >> /etc/dhcp/dhcpd.conf
option domain-name "cluster.rpi";
option domain-name-servers 8.8.8.8, 8.8.4.4;
default-lease-time 600;
max-lease-time 7200;
ddns-update-style none;
authoritative;

subnet 10.10.1.0 netmask 255.255.255.0 {
```

```
range 10.10.1.1 10.10.1.100;
option subnet-mask 255.255.255.0;
option routers 10.10.1.254;
option broadcast-address 10.10.1.255;
host nodo01 {
hardware ethernet b8:27:eb:8f:e1:45;
fixed-address 10.10.1.1;
option host-name "nodo01";
}
host nodo02 {
hardware ethernet b8:27:eb:39:51:97;
fixed-address 10.10.1.2;
option host-name "nodo02";
}
host nodo03 {
hardware ethernet b8:27:eb:dc:12:20;
fixed-address 10.10.1.3;
option host-name "nodo03";
}
host nodo04 {
hardware ethernet b8:27:eb:e5:f7:32;
fixed-address 10.10.1.4;
option host-name "nodo04";
}
host nodo05 {
hardware ethernet b8:27:eb:4d:51:d4;
fixed-address 10.10.1.5;
option host-name "nodo05";
}
host nodo06 {
hardware ethernet b8:27:eb:d9:25:c9;
fixed-address 10.10.1.6;
option host-name "nodo06";
}
host nodo07 {
hardware ethernet b8:27:eb:8b:6d:d1;
```

```
fixed-address 10.10.1.7;
option host-name "nodo07";
}
host nodo08 {
hardware ethernet b8:27:eb:27:83:42;
fixed-address 10.10.1.8;
option host-name "nodo08";
}
host nodo09 {
hardware ethernet b8:27:eb:ad:5f:83;
fixed-address 10.10.1.9;
option host-name "nodo09";
}
host nodo10 {
hardware ethernet b8:27:eb:db:df:4d;
fixed-address 10.10.1.10;
option host-name "nodo10";
}
host nodo11 {
hardware ethernet b8:27:eb:f5:d9:64;
fixed-address 10.10.1.11;
option host-name "nodo11";
}
host nodo12 {
hardware ethernet b8:27:eb:80:09:81;
fixed-address 10.10.1.12;
option host-name "nodo12";
}
host nodo13 {
hardware ethernet b8:27:eb:dc:d7:9e;
fixed-address 10.10.1.13;
option host-name "nodo13";
}
host nodo14 {
hardware ethernet b8:27:eb:f3:a8:7d;
fixed-address 10.10.1.14;
```

```
option host-name "nodo14";
}
host nodo15 {
hardware ethernet b8:27:eb:9d:15:67;
fixed-address 10.10.1.15;
option host-name "nodo15";
}
host nodo16 {
hardware ethernet b8:27:eb:97:03:f4;
fixed-address 10.10.1.16;
option host-name "nodo16";
}
host nodo17 {
hardware ethernet b8:27:eb:88:fb:79;
fixed-address 10.10.1.17;
option host-name "nodo17";
}
host nodo18 {
hardware ethernet b8:27:eb:60:08:b7;
fixed-address 10.10.1.18;
option host-name "nodo18";
}
host nodo19 {
hardware ethernet b8:27:eb:60:88:78;
fixed-address 10.10.1.19;
option host-name "nodo19";
}
}
EOF
```

Como podemos apreciar en el script anterior, el fichero de configuración está dividido en dos partes:

- Parte principal (valores por defecto): especifica los parámetros generales que definen la concesión y los parámetros adicionales que se proporcionarán al cliente.
- Secciones (extienden a la principal)

- Subnet: especifica rangos de direcciones IPs que serán cedidas a los clientes que lo soliciten.
- Host: especificaciones concretas de equipos.

En la parte principal se puede configurar los siguientes parámetros, que más tarde se podrá reescribir en las distintas secciones:

- max-lease-time: tiempo de la concesión de la dirección IP
- default-lease-time: tiempo de renovación de la concesión
- option domain-name-server: se pone las direcciones IP de los servidores DNS que va a utilizar el cliente.
- option domain-name: nombre del dominio que se manda al cliente.
- option subnetmask: subred enviada a los clientes.
- option routers: indicamos la dirección de red de la puerta de enlace que se utiliza para salir a internet.
- option broadcast-address: dirección de difusión de la red.

Al indicar una sección subnet se debe indicar la dirección de la red, la máscara de red, y entre llaves se puede poner los siguientes parámetros:

- range: se utiliza para indicar el rango de direcciones IP que se van a asignar.
- Algunos de los parámetros que se han explicado en la sección principal.

Dentro de la sección host, se pueden generar reservas, lo que permite que a un host siempre se le suministre la misma dirección de IP. Además de especificar el nombre que identifica al host, pueden definirse los siguientes parámetros:

- hardware ethernet: es la dirección MAC de la tarjeta de red del host.
- fixed-address: la dirección IP que se va a asignar.
- option host-name: el nombre de equipo que se va a asignar.
- se puede usar también las opciones ya explicadas en la sección principal.

Antes de reiniciar el servidor y que comience a dar servicio, hay que especificar por cual de las interfaces de red va a trabajar. Se debe editar el archivo /etc/default/isc-dhcp-server, en el cual se modifica el parámetro INTERFACESv4 asignándole el nombre de interfaz (por ejemplo, INTERFACESv4="eth0"). A continuación, se debe reiniciar el servicio ejecutando el comando `systemctl restart isc-dhcp-server.service` para que el mismo tome los cambios realizados.

4.3.4. NFS

Para que OpenMPI funcione, necesita un sistema de archivos al que se pueda acceder desde todos los nodos con la misma ruta. En esta sección se explica cómo configurar un sistema de archivos de red en el nodo de master, el cual será montando y accedido desde los nodos restantes.

NFS es un protocolo de sistema de archivos distribuido que permite montar directorios remotos en un nodo. Esto permite aprovechar el espacio de almacenamiento en una ubicación diferente y escribir fácilmente en él desde otros nodos. Gracias a este servicio exportaremos los directorios `/SOFT` y `/home`, los cuales serán montados en cada nodo Worker permitiendo tener un sólo punto de acceso, lo cual facilitara las tareas a la hora de compilar y ejecutar algoritmos paralelos dentro del cluster.

El Algoritmo 4.3 muestra cómo instalar y configurar el servidor NFS:

Algoritmo 4.3: Bash Script para configurar el servicio NFS.

```
#!/bin/bash
#Instalamos los paquetes
apt install nfs-kernel-server nfs-common -y

#Creamos el directorio donde se alojaran las
#librerías de compilación
mkdir /SOFT

#Cambiamos sus permisos
chown pi:pi /SOFT

#En el archivo /etc/export configuramos los
#directorios que se compartirán en la red
cat << EOF >> /etc/export
```

```
/SOFT 10.10.1.0/255.255.255.0
(rw,sync,no_subtree_check,no_root_squash)
/home 10.10.1.0/255.255.255.0
(rw,sync,no_subtree_check,no_root_squash)
EOF

#Crea en el servidor NFS la tabla de directorios
#compartidos.
exportfs -a

#Reiniciamos el servicio
systemctl restart nfs-kernel-server
```

Lo primero que realiza el algoritmo es instalar los paquetes necesarios para poder utilizar NFS.

nfs-kernel-server: paquete del servidor NFS que nos permitirá compartir los directorios.

nfs-common: paquete que incluye las funcionalidades de NFS (no tiene incluido los componentes de servidor).

Luego se crea el directorio que se utilizará para copiar las librerías que serán usadas para programar los algoritmos paralelos. Este directorio posteriormente se exporta para que el mismo sea montado por todos los nodos worker y tengan acceso a las mismas librerías, evitando tener que replicarlas en cada uno de ellos.

Una vez creado el directorio, modificar el archivo de configuración de NFS para especificar el uso compartido de estos recursos. Básicamente, se debe agregar una línea por cada directorio que se quiere compartir. La sintaxis es la siguiente:

```
directorio_a_compartir        cliente(opción_1,...,opción_N)
```

Resulta conveniente que los directorios estén disponibles en toda nuestra red privada. Como se ve en el algoritmo de configuración se exponen los directorios en toda la red 10.10.1.0/24.

También se puede configurar determinadas características en cada directorio que se comparte. A continuación se explican las utilizadas en este caso:

rw: esta opción le da a la computadora cliente acceso de lectura y escritura al volumen.

sync: esta opción obliga a NFS a escribir cambios en el disco antes de responder. Esto da como resultado un entorno más estable y coherente, ya que la respuesta refleja el estado real del volumen remoto.

nosubtreecheck: esta opción evita la verificación de subárboles, que es un proceso en el que el host debe verificar si el archivo todavía está disponible en el árbol exportado para cada solicitud. Esto puede causar muchos problemas cuando se renombra un archivo mientras el cliente lo tiene abierto. En casi todos los casos, es mejor desactivar la verificación de subárboles.

norootsquash: por defecto, NFS traduce las solicitudes de un usuario root de forma remota a un usuario sin privilegios en el servidor. Se suponía que esto era una característica de seguridad al no permitir que una cuenta root en el cliente usara el sistema de archivos del host como root. Esta directiva deshabilita esto para ciertas acciones.

Ademas de compartir el directorio /SOFT, otro directorio que se comparte es el /home. Este directorio se decidió compartir para evitar la instalación de un servicio como NIS (Network Information System, que en español significa Sistema de Información de Red). NIS permite el envío de datos de configuración en sistemas distribuidos tales como nombres de usuarios y hosts entre dispositivos de una red. Para simplificar la autenticación de usuarios en el cluster, como sólo se utilizará un usuario para autenticarse (*pi*, el usuario por defecto), no se consideró necesario instalar un servicio de esta magnitud. Dado que las RPi son clonadas, lo que garantiza tener el mismo usuario en todas las placas, se optó por compartir el directorio /home y asegurar que todas las placas tendrán también el mismo espacio de usuario.

4.3.5. NAT

Para lograr que los nodos también tengan acceso a Internet, lo cual será necesario sobre todo a la hora de actualizar el sistema operativo o instalar aplicaciones, se deberá configurar el nodo master como un punto de acceso. A diferencia del nodo master, los nodos workers no cuentan con acceso a Internet por su interfaz WIFI. La principal razón para que esto sea así es lograr aislar del exterior los nodos del cluster y sólo interactuar con los mismos a través del nodo master.

Para permitir la navegación a través del nodo master, que es quien tiene una interfaz con conexión a Internet, se debe modificar el parámetro `net.ipv4.ip_forward=0` en el archivo `/etc/sysctl.conf`, y cambiarlo por `net.ipv4.ip_forward=1`. Teniendo habilitado el reenvío de paquetes de una interfaz a otra, ahora es necesario realizar un paso más, habilitar *NAT*.

NAT es un método por el cual se mapean las direcciones IP desde un dominio a otro, en un intento de proporcionar ruteo transparente a las máquinas. Tradicionalmente, los dispositivos de NAT se han usado para conectar un dominio de direcciones privadas no registradas aislado con un dominio externo de direcciones registradas globalmente únicas. Esto garantiza que las máquinas conectadas a la red mediante NAT no son visibles desde el exterior.

Para habilitar NAT en el nodo master se utiliza *Iptables.* IPtables es un sistema de firewall vinculado al kernel de linux. No es un servicio que se inicia, que se suspende o que se puede detener por error; iptables esta integrado con el kernel y es parte del sistema operativo. Es un proceso que aplica reglas. Para ello se ejecuta el comando `iptables`, con el que se crean, modifican o eliminan reglas.

El siguiente comando permite al nodo master compartir la conexión a Internet configurando NAT entre la interfaz wlan0 y la interfaz eth0:

```
iptables -t nat -A POSTROUTING -s 10.10.1.0/24 -o wlan0 -j MASQUERADE
```

Sin embargo, esta regla se debe aplicar cada vez que se inicie el nodo master. Como Raspbian no proporciona una manera de cargar iptables en cada arranque, se necesita agregar manualmente como un script. Iptables-persistent es un paquete de Debian, totalmente compatible con Raspbian. Durante su instalación permite guardar la tabla de iptables actual. Esta operación genera un archivo en /etc/iptables/rules.v4 que contendrá el conjunto de reglas que se hayan creado anteriormente. Este archivo será utilizado por Iptables-persistent para cargar las reglas durante el inicio del sistema. De esta manera se garantiza el acceso a Internet a los nodos del cluster aún si el nodo master es reiniciado.

4.3.6. OpenMPI

El último paso en la configuración del master es la instalación de OpenMPI. Esta instalación se basa en la compilación de los archivos fuentes del mismo; para esto es necesario descargar los archivos de la versión que se utilizará y se procede a compilarlos

en una de las placas Raspberry Pi. Es importante tener en cuenta que los archivos fuentes deben ser compilados en la arquitectura donde serán utilizados.

Para instalar OpenMPI en el nodo master, se debe ejecutar el Algoritmo 4.4 que cuenta con todo lo necesario para llevar a cabo la compilación de los archivos fuentes de la última versión estable:

Algoritmo 4.4: Bash Script para compilar la libreria OpenMPI.

```
#!/bin/bash

#Nos posicionamos en el directorio /tmp
cd /tmp

#Descargamos los archivos fuentes de OpenMPI
wget https://www.open-mpi.org/software/ompi/\
v3.0/downloads/openmpi-3.0.0.tar.bz2

#Descomprimimos el archivo descargado
tar -xf openmpi-3.0.0.tar.bz2

#Accedemos al directorio que se descomprimio.
cd openmpi-3.0.0

#Creamos el archivo build donde se iran compilando
#OpenMPI
mkdir -p build

#Accedemos al directorio build
cd build

#Creamos el directorio donde se instalará
#OpenMPI.
mkdir /SOFT/openmpi

#Le damos permisos al usuario Pi sobre el directorio
chown pi:pi /SOFT/openmpi
```

```
#Ejecutamos configure y preparamos las directivas
#para Make
../configure CFLAGS=-march=armv7-a \
CCASFLAGS=-march=armv7-a \
--prefix=/SOFT/openmpi/ --enable-mpi-fortran \
--disable-mpi-cxx --disable-mpi-cxx-seek \
--disable-java \
--disable-ipv6

#Compilamos e instalamos.
make -j 3 && make install
```

El algoritmo es auto-explicativo, destacándose los dos últimos pasos. El comando `configure` crea un archivo llamado Makefile que ademas de una serie de parámetros propios incluirá aquellos flags pasados por parámetro. El comando `make` realiza la compilación de los archivos fuentes y toma como entrada el archivo generado previamente por el comando configure. El parámetro `-j 3` especifica al comando que debe utilizar 3 núcleos del procesador para compilar, lo que permite acortar los tiempos del proceso de compilación. Para finalizar la instalación, se ejecuta el comando `make install`; el programa toma los archivos binarios de la etapa anterior y los copia en los directorios apropiados para ser utilizados posteriormente. A diferencia de otros sistemas operativos, la instalación sólo requiere la copia de algunas bibliotecas y archivos ejecutables y no hay ningún requisito de registro como tal. En definitiva, `make install` sólo copia los archivos compilados en el destino especificado en el parámetro `-prefix` en el comando `configure`.

Sin embargo, a pesar de tener OpenMPI instalado, es necesario definir tres variables de entorno para poder utilizarlo. Si este paso no se lleva a cabo, las variables no estarán disponibles cuando se compilen los códigos fuentes de las aplicaciones a ejecutar sobre el cluster. Una variable de entorno es un valor dinámico cargado en la memoria, que puede ser utilizado por varios procesos que funcionan simultáneamente. En la mayoría de los sistemas operativos, la ubicación de algunas bibliotecas o de los archivos ejecutables del sistema más importantes puede variar según la instalación. Por eso es posible, para un programa dado, remitirse a una ubicación basada en las variables de entorno que definen estos datos. Para definirlas, editar el archivo */home/pi/.bashrc* y al final del mismo agregar las siguiente lineas:

```
export OPENMPI_PATH=/SOFT/openmpi
```

```
export PATH=$OPENMPI_PATH/bin:$PATH
export LD_LIBRARY_PATH=$OPENMPI_PATH/lib:$LD_LIBRARY_PATH
```

Es necesario hacer un cambio más en el archivo *.bashrc,* en el comienzo del archivo aparece un conjunto de lineas:

```
case $- in
    *i*) ;;
      *) return;;
esac
```

Estas cuatro líneas se deben comentar, para que las mismas no tengan efecto cuando inicia sesión un usuario. Esto se debe a que cuando iniciemos sesión remotamente desde el nodo master a cualquiera de los nodos worker no es recomendable que el mismo sea interactivo ya que podría generar problemas cada vez que MPI envíe un mensaje a un nodo. Se debe recordar que esto se realiza vía SSH y se utiliza autenticación con clave pública para evitar que el sistema operativo solicite las credenciales de autenticación. Anulando estas líneas, se evita que se solicite algún dato al momento de la autenticación e iniciar sesión de forma no interactiva.

Para verificar la correcta instalación de Open MPI se ejecutará el comando `ompi_info` y en la salida del comando se debe visualizar que la versión corresponde a la 3.0.0. En la Figura 4.8 se observa la salida del comando OpenMPI.

Ya se está en condiciones de afirmar que el nodo master se encuentra completamente configurado con los parámetros de red, los servicios necesarios y las librerías de OpenMPI. A continuación se proseguirá con la configuración del nodo worker.

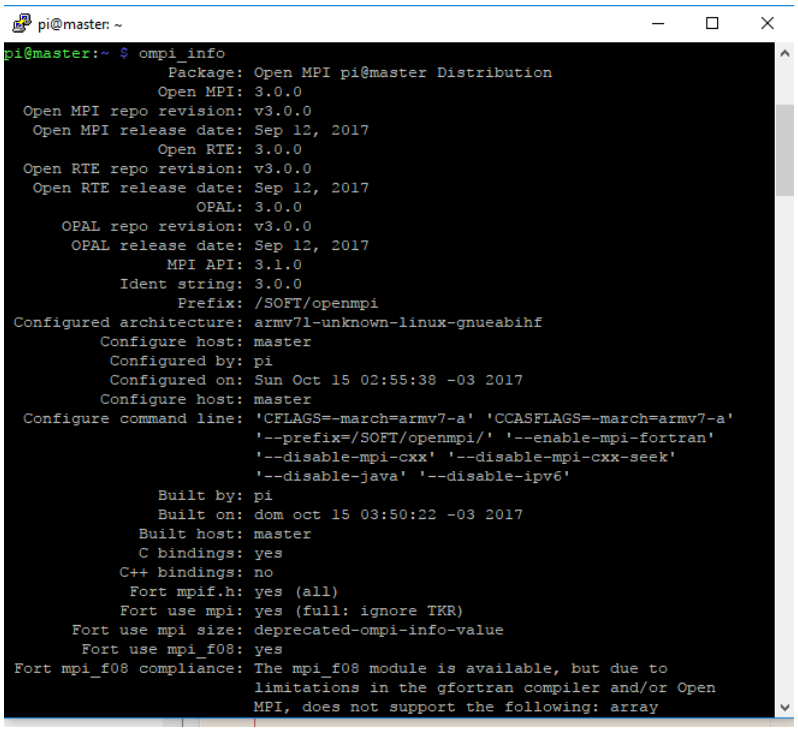


Figura 4.8: Salida del comando *ompi_ info*

4.4. Configuración del nodo worker

A diferencia del nodo master, los nodos worker tienen una configuración más sencilla, en las que se realizan los siguientes pasos:

Configuración de red: configuración de los parámetros mínimos de red para brindar conectividad entre los nodos.

NFS: configuración del cliente NFS y lograr el acceso a los directorios compartidos del nodo master.

4.4.1. Configuración de red

La configuración de este nodo se inicia partiendo de la configuración básica descrita en la sección 4.2.3.

Como se mencionó anteriormente (sección 4.3.5) el acceso a Internet se realiza por medio del nodo master, por lo que el primer paso en la configuración del nodo worker es deshabilitar la interfaz WIFI. Para deshabilitar la interfaz se debe agregar la linea `dtoverlay=pi3-disable-wifi` al final del archivo /boot/config.txt. Para que la configuración surja efecto, es necesario reiniciar la RPi.

El próximo paso es brindar al nodo un nombre el cual permita identificarlo dentro del cluster. Se debe recordar que este nombre será provisto por el servidor DHCP (configurado en el nodo master, en la sección 4.3.3) durante el encendido de la RPi. Sin embargo, para evitar problemas al momento de la asignación automática del nombre del equipo, se debe editar el archivo /etc/hostname, y borrar el contenido de este archivo dejándolo vacío.

Al igual que en el nodo master, es necesario generar el archivo /etc/hosts para poder utilizar los nombres de equipo para referenciarlos dentro de la red del cluster. Se utiliza el Algoritmo 4.1 de la sección 4.3.2 para generar este archivo en cada uno de los nodos worker.

4.4.2. NFS

Para evitar la réplica de archivos en cada uno de los nodos que conforman el cluster, el nodo master comparte un conjunto de directorios que contendrán los archivos necesarios para la ejecución de aplicaciones. De esta manera, se garantiza el acceso de todos los nodos worker a los mismos archivos.

En el Algortimo 4.5 se muestra cómo instalar y configurar el servicio cliente de NFS en el nodo worker:

Algoritmo 4.5: Bash Script para la instalación y configuración de NFS en el nodo Worker.

```
#!/bin/bash

#Instalamos el cliente NFS
sudo apt-get install nfs-common -y

#Creamos el directorio donde se montara el
#directorio /SOFT remoto
mkdir /SOFT

#Modificamos los permisos del directorio
chown pi:pi /SOFT

#Editamos el archivo /etc/fstab
cat << EOF >> /etc/fstab
```

```
master:/SOFT    /SOFT   nfs defaults 0 0
master:/home    /home   nfs defaults 0 0
EOF
```

El script es sencillo. El primer comando instala el cliente NFS y crea el directorio en donde se monta el directorio remoto que exporta el nodo master; otro de los directorios donde se monta un directorio del master es /home, el cual no es necesario crearlo ya que es un directorio que forma parte del sistema de archivos del sistema operativo. Para que el usuario pi pueda hacer uso de estos directorios es necesario proveer los mismos con los permisos pertinentes; para esto es necesario ejecutar los comandos: `chown pi:pi /SOFT` y `chmod 775 /SOFT`.

Por ultimo, el paso más importante del script: la configuración de automontaje de los directorios compartidos, lo que permite que los directorios remotos sean montados durante el arranque de la RPi. De no realizar esta configuración cada vez que un nodo se apague o reinicie, perderá el acceso al directorio compartido. Esto es posible realizarlo gracias al archivo */etc/fstab,* esl cual es usado para definir cómo las particiones, distintos dispositivos de bloques o sistemas de archivos remotos, deben ser montados e integrados en el sistema.

Cada sistema de archivos se describe en una línea separada. Estas definiciones se convertirán con *systemd* en unidades montadas de forma dinámica en el arranque, y cuando se recargue la configuración del administrador del sistema.

El archivo es leído por la orden *mount*, a la cual le basta con encontrar cualquiera de los directorios o dispositivos indicados en el archivo para completar el valor del siguiente parámetro. Al hacerlo, las opciones de montaje que se enumeran en *fstab* también se aplicarán.

Como se aprecia en el script las distintas líneas de configuración presentan la siguiente forma:

```
master:/home    /home   nfs defaults 0 0
```

El archivo /etc/fstab contiene los siguientes campos separados por un espacio o una tabulación:

```
<file system> <dir> <type> <options> <dump> <pass>
```

- `<file system>` - Define la partición o dispositivo de almacenamiento para ser montado.

- `<dir>` - Indica a la orden *mount* el punto de montaje donde la partición (`<file system>`) será montada.
- `<type>` - Indica el tipo de sistema de archivos de la partición o dispositivo de almacenamiento para ser montado. Hay muchos sistemas de archivos diferentes que son compatibles como, por ejemplo: *ext2, ext3, ext4, reiserfs, xfs, jfs, smbfs, iso9660, vfat, ntfs, swap* y *auto*. El *type* auto permite a la orden *mount* determinar qué tipo de sistema de archivos se utiliza. Esta opción es útil para proporcionar soporte a unidades ópticas (CD/DVD).
- `<options>` - Indica las opciones de montaje que la orden mount utilizará para montar el sistema de archivos. Tener en cuenta que algunas opciones de montaje son para sistema de archivos específicos. Algunas de las opciones más comunes son:
  - `auto` - El sistema de archivos será montado automáticamente durante el arranque, o cuando la orden mount -a se invoque.
  - `noauto` - El sistema de archivos no será montado automáticamente, sólo cuando se le ordene manualmente.
  - `ro` - Monta el sistema de archivos en modo sólo lectura.
  - `rw` - Monta el sistema de archivos en modo lectura-escritura.
  - `user` - Permite a cualquier usuario montar el sistema de archivos. Esta opción incluye *noexec, nosuid, nodev*, a menos que se indique lo contrario.
  - `users` - Permite a cualquier usuario perteneciente al grupo *users* montar el sistema de archivos.
  - `nouser` - Sólo el usuario *root* puede montar el sistema de archivos.
  - `owner` - Permite al propietario del dispositivo montarlo.
  - `sync` - Toda E/S se debe hacer de forma sincrónica.
  - `async` - Toda la E/S se debe hacer de forma asíncrona.
  - `defaults` - Asigna las opciones de montaje predeterminadas que serán utilizadas para el sistema de archivos. Las opciones predeterminadas para *ext4* son: *rw, suid, dev, exec, auto, nouser, async.*
- `<dump>` - Utilizado por el programa *dump* para decidir cuándo hacer una copia de seguridad. *Dump* comprueba la entrada en el archivo *fstab* y el número de la

misma le indica si un sistema de archivos debe ser respaldado o no. La entradas posibles son 0 y 1. Si es 0, *dump* ignorará el sistema de archivos, mientras que si el valor es 1, *dum*p hará una copia de seguridad. La mayoría de los usuarios no tendrán *dump* instalado, por lo que deben poner el valor 0 para la entrada `<dump>`.

- `<pass>` - Utilizado por *fsck* para decidir el orden en el que los sistemas de archivos serán comprobados. Las entradas posibles son 0, 1 y 2. El sistema de archivos raíz («root») debe tener la más alta prioridad: 1, todos los demás sistemas de archivos que desea comprobar deben tener un 2. La utilidad *fsck* no comprobará los sistemas de archivos que vengan configurado con el valor 0.

Se debe recordar que es necesario reiniciar el nodo para garantizar que se apliquen todas las configuraciones.

4.5. Pruebas de integración master-worker

Una vez configurado el nodo worker (`nodo01`), se puede conformar un pequeño cluster de dos nodos e iniciar las pruebas de funcionamiento. El fin de estas pruebas es verificar si el nodo master y el nodo worker están correctamente configurados.

En el primer paso se debe iniciar un acceso remoto por SSH al nodo master, o sea, al punto de entrada del cluster. Una vez concretada la conexión, se ingresa el comando:

```
ping nodo01
```

Si se obtiene una salida como la de la Figura 4.9, se concluye en que la interfaz Ethernet del nodo worker fue correctamente configurada por el servidor DHCP.

Con la interfaz del nodo worker configurada y funcionando, se procede a verificar el funcionamiento del servicio SSH de este mismo nodo. Desde la terminal del nodo master ejecutar el siguiente comando:

```
ssh nodo01
```

Ante la solicitud de usuario y contraseña, ingresar las credenciales por defecto. Si la conexión es satisfactoria se obtendrá como respuesta una ventana como la que muestra la Figura 4.10. También se puede apreciar que el nombre del equipo fue asignado correctamente.

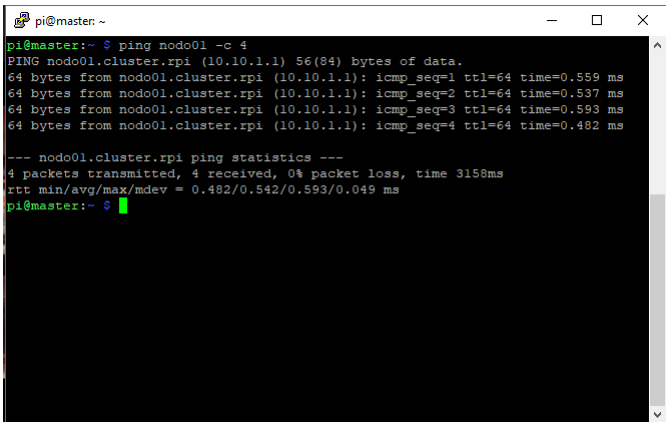


Figura 4.9: Pruebas de funcionamiento: Ping a *nodo01*.

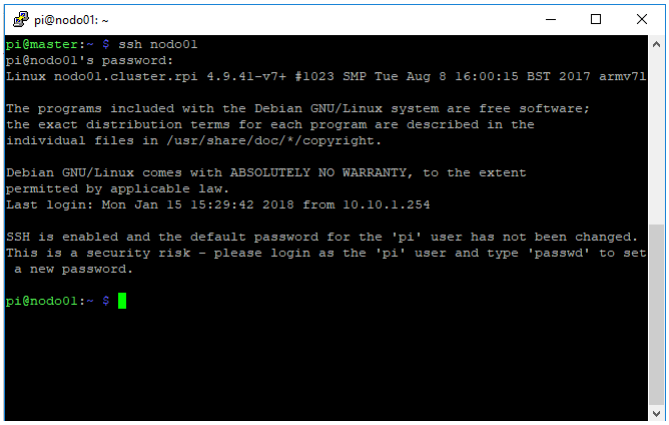


Figura 4.10: Pruebas de funcionamiento: Conexión SSH a *nodo01*.

Como se mencionó anteriormente, si no se utiliza la autenticación con clave publica, las comunicaciones que genere la librería OpenMPI entre el nodo master y los nodos worker sera imposible de establecer ya que la librería no cuenta con la lógica necesaria para manejar credenciales del tipo usuario y contraseña. Para solucionar este problema se utiliza la autenticación por clave publica. El siguiente paso será agregar la clave publica del usuario pi (la cual se generó anteriormente en la sección 4.3.1) al listado de claves autorizadas para el ingreso al sistema. Se debe ejecutar el siguiente comando para copiar la clave publica del nodo master al nodo worker:

```
ssh-copy-id nodo01
```

Durante la ejecución del comando se solicita el ingreso de la contraseña del usuario pi para corroborar la operación y realizar la copia de la clave al `nodo01`. Si la operación se concreta se obtendrá una salida como la de la Figura 4.11.

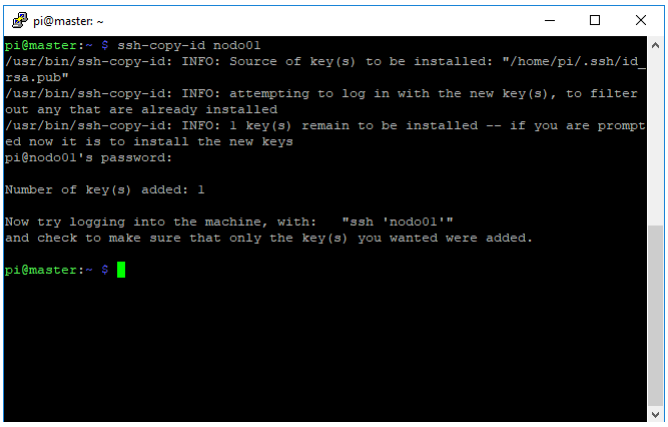


Figura 4.11: Pruebas de funcionamiento: Copia de clave pública.

El siguiente paso consiste en verificar que la clave se copió correctamente ejecutando el comando `ssh nodo01`. Se debe notar que en esta oportunidad no fue necesario ingresar las credenciales de acceso para establecer la conexión.

A continuación, desde el nodo worker, se verifica si los directorios compartidos están accesibles. Se debe ejecutar el comando `df -h` para obtener un listado de los directorios montados en el sistema. En la Figura 4.12, se puede apreciar los directorios `/SOFT` y `/home` exportados del nodo master (`master:`) se encuentran montados y accesibles.

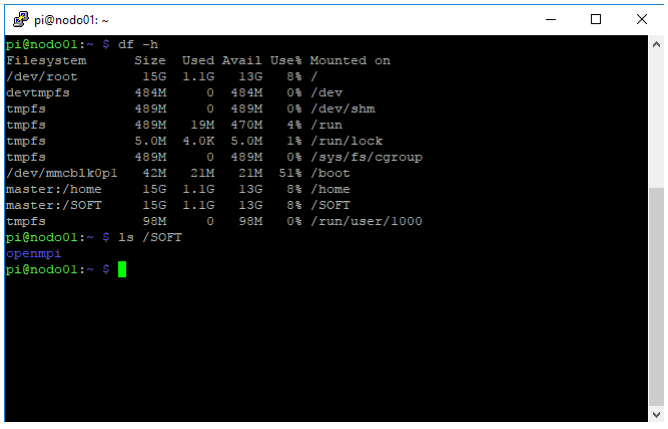


Figura 4.12: Pruebas de funcionamiento: Directorios compartidos.

4.6. Pruebas del cluster

Para configurar los nodos restantes, se realiza una copia de la MicroSD del nodo worker, y se utiliza esta imagen para clonarla en las otras tarjetas. Una vez que todos los nodos fueron replicados, es necesario conectar los mismos al switch por medio de

su interfaz Ethernet y a la alimentación.

Una vez encendidos, se procede a verificar el correcto funcionamiento del cluster. Para esto se cuenta con una pequeña aplicación que utiliza la librería OpenMPI. El Algoritmo 4.6 se utilizará para realizar esta prueba.

Algoritmo 4.6: Algoritmo para probar el funcionamiento del cluster: mpi_hola_mundo.c

```
#include <mpi.h>
#include <stdio.h>
int main(int argc, char** argv) {
// Inicializar el entorno MPI
MPI_Init(NULL, NULL);

// Obtener la cantidad de procesadores
int world_size;
MPI_Comm_size(MPI_COMM_WORLD, &world_size);

// Obtener el rank del proceso
int world_rank;
MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

// Obtener el nombre del procesador
// processor_name[MPI_MAX_PROCESSOR_NAME];
int name_len;
MPI_Get_processor_name(processor_name, &name_len);

// Imprimir el mensaje de Hola mundo
printf("Hola mundo desde el procesador %s, rank %d" " de %d
procesadores\n", processor_name, world_rank, world_size);

// Finalizar el entorno MPI.
MPI_Finalize();
}
```

El siguiente paso es compilar el algoritmo `mpi_hola_mundo.c`. Para compilar, OpenMPI provee el comando `mpicc`, al cual se le especifica como parámetro el archivo del algoritmo y el nombre del archivo resultado. Se debe ejecutar el siguiente comando

para compilar los archivos fuentes de la aplicación de prueba:

```
mpicc mpi_hola_mundo.c -o mpi_hola_mundo
```

Como salida de este comando se obtiene un archivo ejecutable de la aplicación.

Antes de ejecutar la aplicación `mpi_hola_mundo`, es necesario crear un archivo de maquinas el cual es requerido por OpenMPI. Un archivo de máquina contiene una lista de los nodos posibles en los que se desea que se ejecute la aplicación MPI. Entonces, se crea el archivo `machinefile` y en su interior se definen las siguientes lineas:

```
nodo01 slots=4
nodo02 slots=4
nodo03 slots=4
nodo04 slots=4
nodo05 slots=4
nodo06 slots=4
nodo07 slots=4
nodo08 slots=4
nodo09 slots=4
nodo10 slots=4
nodo11 slots=4
nodo12 slots=4
nodo13 slots=4
nodo14 slots=4
nodo15 slots=4
nodo16 slots=4
nodo17 slots=4
nodo18 slots=4
nodo19 slots=4
```

Cada línea tiene la forma `<nombre_de_equipo> <slots=cantidad_núcleos>`. `<nombre_de_equipo>` es el nombre del nodo donde se enviaran procesos, y `<slots=cantidad_núcleos>` es la cantidad de unidades de procesamiento disponibles que tiene ese nodo. Cada procesador de RPi tiene cuatro núcleos disponibles, esto quiere decir que se pueden asignar, como máximo y al mismo tiempo, cuatro procesos a cada nodo (aunque asignar más procesos por placa es posible, no todos los procesos podrían ejecutarse al mismo tiempo).

El siguiente paso es ejecutar la aplicación, para eso se utiliza el comando `mpirun` de la siguiente manera:

```
mpirun -np 72 --hostfile machinefile mpi_hola_mundo
```

Si la ejecución es correcta, se obtiene una salida sin mensajes de error e igual a la que se observa en la Figura 4.13.

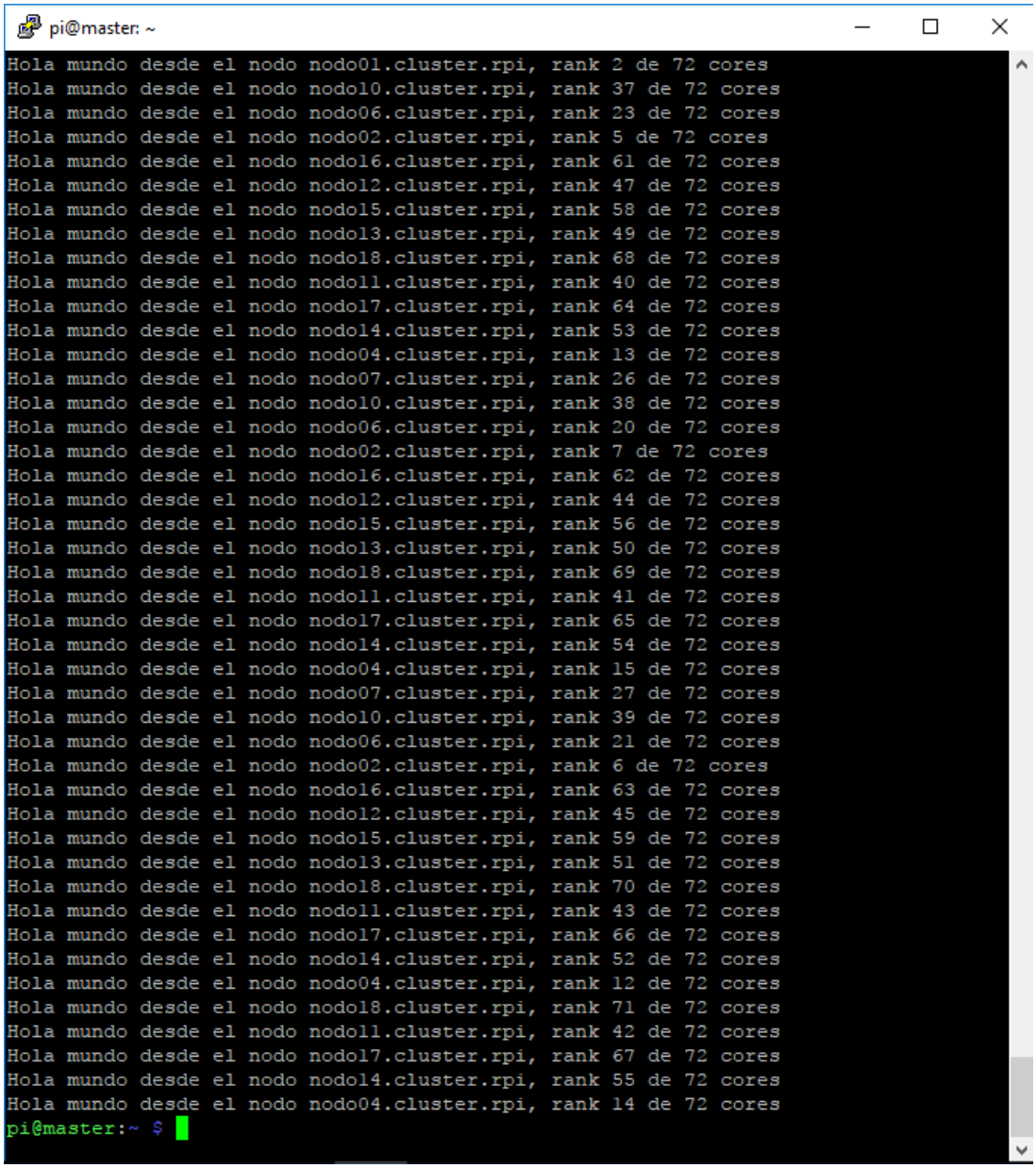


Figura 4.13: Resultado de la ejecución de la aplicación: mpi_hola_mundo

4.7. Resumen

Gracias al bajo costo y al reducido tamaño de las placas RPi, construir un cluster para explorar el mundo de la computación paralela hace que esto sea accesible y fácil de llevar a cabo para un usuario con pocos conocimientos sobre el tema. Si bien las RPi no son la opción adecuada para un sistema complejo de producción, resulta interesante el conjunto de herramientas que brinda para aprender las tecnologías que conforman los cluster profesionales.

Un cluster está formado por un conjunto de dispositivos conectados entre sí para que se puedan comunicar. Para realizar el despliegue del mismo se requieren una serie de elementos: los dispositivos encargados de realizar el cómputo y el control (nodos); un dispositivo que brinde un medio de comunicación que permita interconectar los mismos; y una fuente de poder que permita satisfacer la demanda energética de los dispositivos de cómputo. Para que los dispositivos se comuniquen entre sí se debe configurar el hardware de red y realizar las conexiones de las placas al mismo por medio de cables.

Para el armado del cluster presentado en esta tesina, se necesitó de: 20 placas RPi 3 Model B, un switch 3COM 3C16475BS, 20 tarjetas SD, 20 cables Ethernet/RJ45, una fuente de PC de al menos 250W, un teclado USB, una laptop o PC de escritorio y una conexión a internet.

Como configuración base para las placas se instaló el sistema operativo Raspbian Stretch Lite; y por medio de una conexión WIFI, se actualizaron los paquetes del sistema operativo a sus últimas versiones. Luego, utilizando el asistente de configuración de la RPi se modificaron una serie de parámetros a fin de aumentar el rendimiento de las placas.

El nodo master es el equipo desde el que se administra el cluster. En él se llevan a cabo tareas como lanzamiento de trabajos, instalación de librerías y programas, obtención de los resultados de las ejecución de las aplicaciones, etc. Además, es el punto de acceso al resto de los equipos que conforman el cluster. Para esto se necesita realizar una serie de configuraciones e instalar algunos servicios que permitan al nodo administrar el cluster, como por ejemplo: generación de claves para autenticación, configuración de los parámetros de red, servicios de red (DHCP, NFS, NAT) e instalacion de OpenMPI.

La configuración de los nodos worker es mas sencilla: sólo se necesita realizar las configuraciones de red pertinentes e instalar y configurar el servicio de NFS para acceder a los directorios compartidos provistos por el nodo master.

Una vez configurados los nodos y conectados al switch, es necesario realizar un conjuntos de pruebas a fin de garantizar la integración de los mismos. Para esto se verifica que la autenticación por clave pública esté funcionando correctamente realizando conexiones entre los nodos y comprobando que no se solicita autenticación para establecer la conexión remota. Otra de las pruebas consiste en verificar que los directorios compartidos se encuentran accesibles desde todos los nodos worker.

Realizadas las pruebas de integración se procede a ejecutar pruebas para verificar el correcto funcionamiento del cluster. Esto se logra con una pequeña aplicación (mpi_hola_mundo) que utiliza la librería OpenMPI, ejecutando la misma en todos los nodos del cluster, analizar su salida y verificar que esta sea correcta.

Capítulo 5

Evaluación de rendimiento y eficiencia energética

En este capítulo se detallan y describen el conjunto de pruebas que se utilizaron para evaluar el rendimiento del clúster presentado en el Capítulo 4. Para ello se seleccionaron 3 aplicaciones que se pueden considerar clásicas y representativas de aplicaciones científicas, que poseen alta demanda computacional pero con diferentes características entre sí. Para evaluar el cluster de RPi, se analizaron las prestaciones de las aplicaciones paralelas seleccionadas. En forma complementaria, se compararon con los resultados obtenidos en un cluster de multicores x86 considerando no sólo el rendimiento sino también la eficiencia energética.

Multiplicación de matrices es una aplicación masivamente paralela que demanda gran cantidad de cómputo regular. Requiere poca cantidad de comunicaciones al inicio y al final de cada tarea, pero de gran tamaño.

Jacobi solver es una aplicación regular con alto grado de paralelismo y de dependencia a nivel de datos entre las distintas tareas y etapas de ejecución. Esta gran cantidad de sincronización entre las tareas genera una considerable cantidad de comunicaciones de moderado tamaño a lo largo de toda la ejecución de la aplicación.

N-reinas es una aplicación con alto grado de paralelismo, pero las tareas son irregulares y de gran tamaño, lo que obliga a realizar la distribución dinámica de esas tareas entre los procesos. Esto genera una gran cantidad de comunicaciones no estructuradas de tamaño pequeño.

5.1. Implementación de aplicaciones benchmark

Esta sección se enfoca en el análisis de cada una de las pruebas. Es necesario conocer en detalle el funcionamiento y comportamiento de estas a fin de lograr una mejor interpretación del conjunto de resultados obtenidos. A continuación se profundiza cada una de ellas.

5.1.1. Multiplicación de matrices

Desde los principios de la computación paralela, la multiplicación de matrices es uno de los problemas numéricos que más se ha estudiado, debido a que es una de las operaciones fundamentales del álgebra lineal. En particular, la multiplicación de matrices es una parte esencial de varios algoritmos conocidos para otros problemas computacionales de álgebra lineal y combinatoria, tales como la solución de un sistema de ecuaciones lineales, la inversión de una matriz, la evaluación de la determinante de una matriz, entre otros. Además, el tiempo requerido para la multiplicación de matrices suele ser la parte dominante del tiempo total requerido para la computación de todos esos problemas. Esto significa que al acelerar la multiplicación de matrices, se está reduciendo el tiempo de ejecución de todos los problemas de los cuales es parte.

El problema de multiplicación de matrices es un excelente candidato para paralelización, debido a dos características que este problema tiene: el patrón de acceso a los datos de las matrices y la independencia en los cálculos. Para el cálculo de $C = A \times B$, las matrices A y B se acceden sólo para lectura y la matriz C es la única sobre la que habría que coordinar el acceso de escritura en un ambiente de memoria compartida (es el caso de los cuatro núcleos de una misma placa RPi 3).

La multiplicación de matrices tiene características muy específicas en lo que se refiere al diseño e implementación de un algoritmo paralelo:

- Cada elemento de C, c_{ij}, es independiente de todos los demás elementos. Esta independencia es un factor clave en lo referente a la paralelización.
- La cantidad y el tipo de operaciones a realizar es independiente de los datos mismos.
- Los datos están organizados en estructuras bidimensionales (las matrices mismas) y las operaciones son básicas, de multiplicación y suma.

Algunas de las aplicaciones en donde se aplica esta clase de problemas son:

- Resolución de problemas de cálculo numérico, por ejemplo: resolución de ecuaciones [33].
- Procesamiento de señales [34].
- Se utiliza en el cálculo de microarrays, en el área de bioinformática [35].

Descripción del problema

Dadas dos matrices A y B de dimensiones $m \times n$ y $n \times p$, respectivamente, se define su producto $A \times B$ como la matriz de dimensión $m \times p$ tal que el elemento de la posición fila i y columna j es el resultado del producto punto fila i de A y columna j de B.

Matemáticamente, si las matrices son

$$A = (a_{i,j})_{1 \leq j \leq n}^{1 \leq i \leq m}$$

$$B = (b_{i,j})_{1 \leq j \leq p}^{1 \leq i \leq n}$$

entonces el producto $A \times B$ es

$$A \times B = C$$

$$A \times B = (c_{i,j})_{1 \leq j \leq p}^{1 \leq i \leq m}$$

siendo

$$c_{i,j} = \sum_{k=1}^{n} a_{i,k} \times b_{k,j}$$

Solución secuencial

Algoritmo 5.1 Multiplicación de matrices: pseudocódigo de la solución secuencial.

```
## TB: tamaño de bloque
## n: tamaño de matriz
main {

      inicializar(A);
      inicializar(B);
      inicializar(C);
      multiplicacion_x_bloque(A,B,C,n,TB);

}
multiplicacion_x_bloque (A,B,C,n,TB) {

      for (i:0..n - 1;i += TB) {
           for (j:0..n - 1;j += TB) {
                for  (k:0..n - 1;k += TB) {

                   multiplicar_bloque(A[i*n + k],B[j*n + k],C[i*n + j],n,TB);
                }
           }
      }

}
multiplicar_bloque (bloque_A,bloque_B,bloque_C,n,tb) {

      for (i:0..tb - 1) {
           for (j:0..tb - 1) {
                for  (k:0..tb - 1) {
                   C[i*n + j] += A[i*n + k] * B[j*n + k];
                }
           }
      }

}
```

Para la solución secuencial se optó por utilizar el algoritmo de multiplicación de matrices por bloques. La idea general de esta solución es organizar las estructuras de datos en un programa en partes más pequeñas llamadas bloques.

El programa está diseñado para que cargue un bloque del tamaño de la cache, haga todas las lecturas y escrituras que necesite en esa porción, luego lo descarte, cargue la siguiente porción, y así sucesivamente. Si bien esta técnica complejiza la solución y dificulta la comprensión del algoritmo, es interesante para aplicar ya que puede producir grandes ganancias de rendimiento en algunos sistemas al aprovechar la localidad de los datos.

Un algoritmo de multiplicación de matrices por bloque funciona dividiendo las matrices en submatrices y luego explotando el hecho matemático de que estas submatrices pueden manipularse como los escalares. Por ejemplo, supongamos que queremos

calcular $C = AB$, donde A, B y C son matrices de 8×8. Entonces podemos dividir cada una en cuatro submatrices de 4×4:

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

donde:

$$C_{11} = A_{11}B_{11} + A_{12}B_{21}$$

$$C_{12} = A_{11}B_{12} + A_{12}B_{22}$$

$$C_{21} = A_{21}B_{11} + A_{22}B_{21}$$

$$C_{22} = A_{21}B_{12} + A_{22}B_{22}$$

La idea básica detrás de este código es dividir A,B y C en bloques de $TB \times TB$ elementos. Entonces, mientras que la rutina *multiplicación_x_bloque* itera sobre las submatrices identificadas, la rutina *multiplicar_bloque* itera sobre los elementos de las submatrices A y B resolviendo la multiplicación del bloque en sí, almacenando el resultado obtenido en el bloque C.

Solución paralela

La mayoria de los algoritmos paralelos para la multiplicación de matrices usan la descomposición de matrices que se basan en la cantidad de núcleos disponibles. Esto incluye la descomposición en bloques de columnas o filas. Durante la ejecución, un proceso calcula un resultado parcial utilizando las submatrices que le fueron asignadas en un momento dado. Luego sucesivamente este proceso realiza el mismo cálculo en nuevas submatrices, añadiendo el nuevo resultado al anterior. Cuando todos los procesos completan la multiplicación de los distintos bloques, el proceso principal o master ensambla los resultados parciales y genera el resultado total de la multiplicación de matrices.

Los algoritmos propuestos para la multiplicación de matrices en paralelo suelen seguir el modelo de cómputo SPMD [36] [37]. De esta manera, un mismo programa se ejecuta de forma asincróna en cada núcleo de la máquina paralela y eventualmente se sincroniza y/o comunica con los demás núcleos.

Cada proceso calcula una porción de C y para eso requiere contar con los datos necesarios para hacerlo. Una vez que cada proceso tiene asignada una submatriz de

C, en función a esta última se deben distribuir los datos de las matrices A y B. La Figura 5.1 ilustra lo mencionado anteriormente.

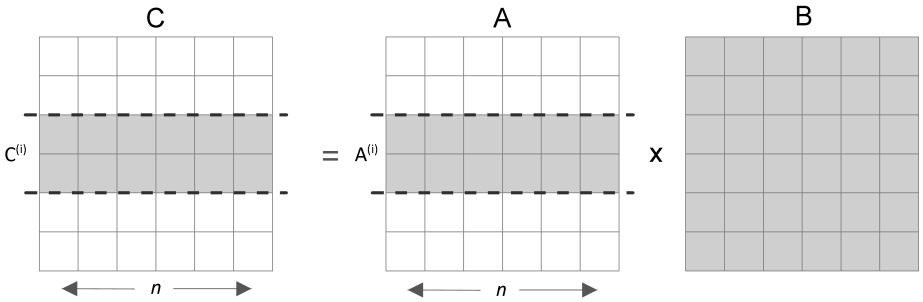


Figura 5.1: Cálculo de la submatriz.

Observando la figura, es simple darse cuenta que cada proceso de la solución debe tener localmente los datos de la submatriz$A^{(i)}$, para poder llevar a cabo el cálculo de $C^{(i)}$. También para llevar adelante el cálculo, resulta necesario tener una copia de B en cada uno de los procesos. De esta manera, es posible llevar a cabo en simultáneo los cálculos de los distintos $C^{(i)}$debido a que todos los procesos cuenta con lo necesario para poder hacerlo.

Para implementar las comunicaciones, se utilizaron las operaciones de comunicación colectiva de MPI: MPI_Scatter para la distribución de A, MPI_Bcast para el envío de B y MPI_Gather para la recuperación de C.

Dicho esto, en el Algoritmo 5.2 se muestra el pseudocódigo de la solución paralela de una multiplicación de matrices por bloque.

Algoritmo 5.2 Multiplicación de matrices: pseudocódigo de la solución paralela.

```
## ntasks: cantidad de procesos
## strip: tamaño de bloque
## n: tamaño de matriz
process worker [0..ntasks] {

      strip = n / ntasks;
      if (myrank == MASTER) {
           A = array(n*n);
           B = array(n*n);
           C = array(n*n);


           inicializar_matriz(A);
           inicializar_matriz(B);
      } else {
           A = array(strip*n);
           B = array(n*n);
           C = array(strip*n);
      }
      ## Enviar una porción de a a cada proceso;
      Scatter (A,strip*n,MASTER);
      ## Enviar B a todos los procesos;
      Bcast(B,n*n);
      multiplicacion_x_bloque (A,B,C,n,tm,tb);
      ## Recibir todos los resultados
      Gather(C,strip*n,MASTER);

}
```

5.1.2. Jacobi-solver

El método Jacobi es un método iterativo para resolver sistemas de ecuaciones lineales y se aplica sólo a sistemas cuadrados, es decir a sistemas con tantas incógnitas como ecuaciones. Este tipo de iteración es ampliamente usada en diferentes problemas por su sencillez en la implementación y por su potencial capacidad para ser paralelizada. El método de Jacobi tiene aplicaciones en áreas como:

- Resolución de ecuaciones lineales que describen fenómenos físicos como el flujo de fluidos, la electrostática, el clima, flujo de aire sobre un viento, entre otros [38].
- Biometría [39].
- Biología computacional [40].
- Procesamiento digital de señales [41].
- Procesamiento de imágenes [42].

Descripción del problema

El método de Jacobi es utilizado como base para muchos métodos iterativos del análisis numérico y la simulación. En su forma más directa, se puede usar para resolver la ecuación de difusión para una función escalar $\Phi(\vec{r}, t), \frac{\partial \Phi}{\partial t} = \Delta\Phi$, en una matriz cuadrada sujeta a las condiciones de contorno de Dirichlet. Los operadores diferenciales se discretizan utilizando diferencias finitas:

$$\frac{\delta\Phi(x_i, y_i)}{\delta t} = \frac{\Phi(x_i + 1, y_i) + \Phi(x_i - 1, y_i) - 2\Phi(x_i, y_i)}{(\delta x)^2} + \frac{\Phi(x_i, y_i - 1) + \Phi(x_i, y_i + 1) - 2\Phi(x_i, y_i)}{(\delta y)^2} \tag{5.1}$$

.

En cada paso de tiempo, una corrección de $\delta\Phi$ a Φ en la coordenada (x_i, y_i) se calcula mediante 5.1 utilizando los valores antiguos de los cuatro puntos vecinos siguientes. Por supuesto, los Φ valores actualizados deben escribirse en una segunda matriz. Después de que todos los puntos hayan sido actualizados (un "barrido"), el algoritmo se repite [43].

Solución secuencial

La idea general de la solución secuencial, consiste en llevar a cabo iteraciones en una matriz de lectura y escribiendo el resultado en una matriz de escritura. Al final de cada iteración es necesario intercambiar las matrices. Por lo tanto, es necesario contar con el doble de memoria para poder mantener dos copias del volumen de datos.

Por otra parte, el nuevo valor para cada elemento de la matriz se establece en el promedio de los valores anteriores de los puntos a la izquierda, a la derecha, arriba y debajo de él (ver Figura 5.2). Este proceso se repite una determinada cantidad de iteraciones (configurable). La matriz es de tamaño $n \times n$ y se encuentra rodeada por un cuadrado inicializada con valores límite (ver Figura 5.3a).

Como puede apreciarse en el Algoritmo 5.3, se utilizan dos matrices para intercambiar los resultados en cada una de las iteraciones de la ejecución. Para poder calcular los valores de una matriz en una iteración, se utilizan los valores de la otra, y así sucesivamente hasta completar la totalidad de la iteraciones. Antes de realizar cualquier tipo de operación sobre las matrices es importante que los bordes de ambas matrices contengan los valores límite apropiados, y que los puntos interiores se inicialicen con algún valor por defecto. Por ultimo, se calcula la diferencia máxima entre ambas matrices.

Cabe destacar que, por una cuestión de eficiencia, se representan las matrices

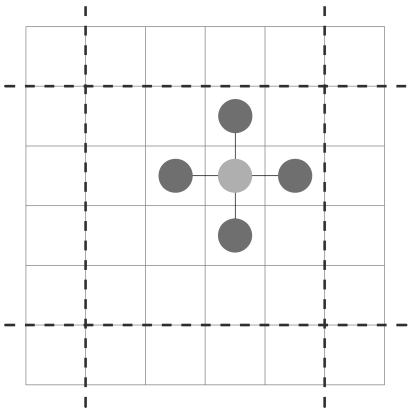


Figura 5.2: Jacobi solver: cálculo del nuevo valor.

$$\begin{bmatrix} 100 & 100 & 100 & 100 & 100 & 100 \\ 30 & 51 & 51 & 51 & 51 & 70 \\ 30 & 51 & 51 & 51 & 51 & 70 \\ 30 & 51 & 51 & 51 & 51 & 70 \\ 30 & 51 & 51 & 51 & 51 & 70 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(a) Ejemplo de matriz cuadrada de 4x4 con valores límites.

100	100	100	100	100	100	30	51	51	51	51	70	30	51	51	51	51	70
(0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)	(15)	(16)	(17)
30	51	51	51	51	30	70	51	51	51	51	70	0	0	0	0	0	0
(18)	(19)	(20)	(21)	(22)	(23)	(24)	(25)	(26)	(27)	(28)	(29)	(30)	(31)	(32)	(33)	(34)	(35)

(b) Matriz de 4x4 representada como un arreglo.

Figura 5.3: Representación de las estructuras de datos utilizadas en las pruebas de Jacobi solver.

(estructuras de dos dimensiones) como vectores (estructuras de una dimensión). En la Figura 5.3b puede apreciarse la representación de la matriz como un arreglo.

Con respecto al algoritmo utilizado para esta solución, es importante aclarar que el mismo se encuentra optimizado. El conjunto de optimizaciones aplicadas son enumeradas a continuación:

1. se reemplazaron todas las operaciones de división por operaciones de multiplicación, ya que son más eficientes a la hora de ser ejecutadas.

2. se reemplazaron todas las variables de tipo matrices (bidimensionales) por variables de tipo arreglos (unidimensionales), este cambio permite que el manejo de memoria sea mas eficiente.

3. se desenrolló el bucle principal para incluir 2 fases de actualización por iteración (una para *old*, otra para *new*). Esta técnica permite disminuir la cantidad de instrucciones que controlan el bucle.

Algoritmo 5.3 Pseudocódigo del algoritmo secuencial optimizado para resolver la iteración de Jacobi

```
## MAXITER: cantidad de iteraciones
## n: tamaño de la matriz
real old = array(n+1,n+1);
real new = array(n+1,n+1);
real maxdif= 0.0;
inicializar(old);
inicializar(new);
for (iter: 1..MAXITERS,i+=2) {

      # calcular los nuevos valores de todos los elementos interiores
      # en funcion a los valores de los elementos vecinos
      for (i:1..n,j:1..n) {

            real vsup = old[i-1,j], vinf = old[i+1,j], vizq = old[i,j-1], vder = old[i,j+1];
            new[i,j]= (vsup + vinf + vizq + vder) * 0.25;
      }
      # calcular nuevamente los valores de todos los elementos interiores
      for (i:1..n,j:1..n) {

            real vsup = new[i-1,j], vinf = new[i+1,j], vizq = new[i,j-1], vder = new[i,j+1];
            old[i,j]= (vsup + vinf + vizq + vder) * 0.25;
      }

}
# calcular la diferencia máxima
for (i:1...n,j:1..n)

      maxdif = max(maxdif, abs(old[i,j]-new[i,j]))
```

Solución paralela

Algoritmo 5.4 Pseudocódigo del mejor algoritmo paralelo para resolver la iteración de Jacobi.

```
process worker [w:0..ntasks] {

      int strip = n/ntasks;
      real old = array((strip+2)*(n+2));
      real new = array((strip+2)*(n+2));
      inicializar(old);
      inicializar(new);
      for (iter:1..MAXITERS,i+=2) {
            # Actualización de new a partir de old.
            for (i:1..strip,j:1..n) {

                  real vsup = old[i-1,j], vinf = old[i+1,j], vizq = old[i,j-1], vder = old[i,j+1];
                  new[i,j]= (vsup + vinf + vizq + vder) * 0.25;
            }
            # Comunicación de los bordes (2 send de new, 2 recv en new).
            if (w > 1)
                  Send (w-1,new+n,n+2);
            if (w < ntasks)
                  Send (w+1,new+strip*n,n+2);
            if (w < ntasks)
                  Recv(w+1,new+(strip+1)*n,n+2);
            if (w > 1)
                  Recv(w-1,new,n+2);
            # Actualización de old a partir de new.
            for (i:2..strip,j:1..n) {

                  real vsup = new[i-1,j], vinf = new[i+1,j], vizq = new[i,j-1], vder = new[i,j+1];
                  old[i,j]= (vsup + vinf + vizq + vder) * 0.25;
            }
            # Comunicación de los bordes (2 send de old, 2 recv en old);
            if (w > 1)
                  Send(w-1,old+n,n+2);
            if (w < ntasks)
                  Send(w+1,old+strip*n,n+2);
            if (w < ntasks)
                  Recv(w+1,old+(strip+1)*n,n+2);
            if (w > 1)
                  Recv(w-1,old,n+2);
      }
      # Calcular la diferencia máxima de mi strip.
      for (i:1..HEIGHT,j:1..strip) {
            mi_dif = max(mi_dif, abs(old[i,j]-new[i,j]);
      Reduce(mi_dif, MAX, 0)
      # El valor de diferencia esta guardado en mi_dif del worker 0

}
```

Según el pseudocódigo del Algoritmo 5.4, se pueden utilizar *ntasks* procesos y hacer que cada uno actualice un conjunto de valores de la matriz *old*. La matriz se

distribuye entre todos los procesos asignando un bloque de filas a cada uno de ellos. También se deben replicar las filas en los bordes de los strips, porque los procesos workers no pueden leer datos locales de otros procesos. En particular, cada proceso worker necesita almacenar no sólo los elementos en su strip, sino también los elementos en los bordes de los strips adyacentes. Cada proceso worker ejecuta una secuencia de fases de actualización; después de cada actualización, intercambia los bordes de su strip con sus vecinos (ver Figura 5.4).

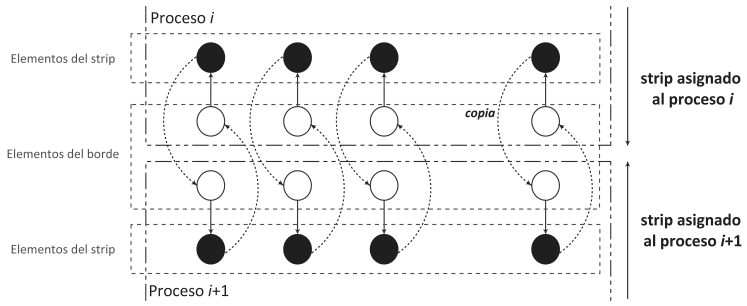


Figura 5.4: Fase de distribución en la solución paralela de Jacobi-solver.

Se puede apreciar que en la solución presentada los procesos workers vecinos intercambian bordes dos veces por iteración. El primer intercambio se programa como:

```
if (w > 1) Send (w-1, new+n, n+2);
if (w < ntasks) Send (w+1, new+strip*n, n+2);
if (w < ntasks) Recv(w+1, new+(strip+1)*n, n+2);
if (w > 1) Recv(w-1, new, n+2);
```

Todos los procesos worker excepto el primero envían la fila superior de su strip al vecino de arriba; todos los procesos worker, excepto el último, envían la fila inferior de su strip al vecino de abajo. Cada proceso recibe los bordes de sus vecinos; estos se convierten en los límites de su strip. El segundo intercambio es idéntico, excepto que se usa la matriz *old* en lugar de *new*.

Después de un número apropiado de iteraciones, cada worker calcula su diferencia máxima local. La diferencia máxima final se computa mediante una operación colectiva de reducción.

5.1.3. N-reinas

Esta aplicación pertenece a una clase de problemas en los que el tiempo de comunicación entre procesos Tc no es significativo frente al tiempo de procesamiento local Tp

(Tp >> Tc). Además, debido a su complejidad computacional de orden exponencial, se hace difícil procesar tableros.

El problema de N-reinas suele verse como puramente matemático y recreativo, aunque posee varias aplicaciones en el mundo real [44]:

- Control de tráfico aéreo.
- Compresión de datos.
- Balance de carga y ruteo de mensajes o datos en un multiprocesador.
- Prevención de deadlocks.
- Administración de almacenamientos de memorias compartidas.
- Procesamiento de imágenes.
- Detección de movimientos en una red distribuida.

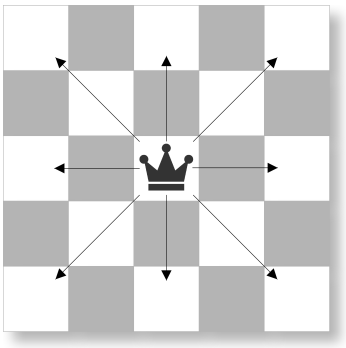
Descripción del problema

El problema de las N-reinas es una generalización al problema de las 8-Reinas (problema combinatorio). Este problema consiste en ubicar en un tablero de ajedrez estándar (8×8) ocho reinas de tal forma que no se amenacen entre sí. La reina tiene la particularidad que es capaz de desplazarse por el tablero la cantidad de casillas que se desee y puede hacerlo en forma vertical, horizontal y diagonal, como se muestra en la Figura 5.5a. Por esta razón, no pueden convivir dos reinas en una misma columna, fila o diagonal. En la Figura 5.5b se aprecia una de las ocho posibles soluciones del problema en un tablero de 5×5. En cuanto al problema de las N-reinas, el mismo consiste en determinar cuantas maneras diferentes hay de ubicar N reinas en un tablero de $N \times N$, para $N > 0$ [45][46].

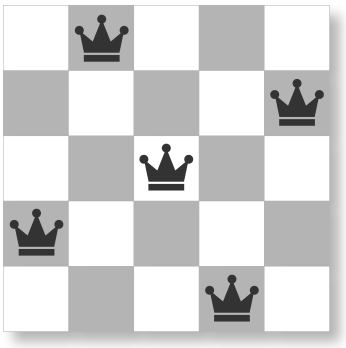
Una solución inicial al problema de las N-reinas, mediante un algoritmo secuencial sencillo (basado en fuerza bruta), es quedarse con las combinaciones validas que surgen de probar todas las posibles ubicaciones de las reinas en un tablero.

Solución secuencial

En esta sección se realiza una descripción del algoritmo secuencial para resolver el problema en un tablero $N \times N$. Utilizar un algoritmo de fuerza bruta que evalúe todas las combinaciones posibles sería muy costoso, de manera que se utiliza un algoritmo



(a) Movimientos de la pieza de la reina en un tablero de 5×5.



(b) Solución en un tablero de 5×5.

Figura 5.5: Problema de las N-reinas.

con optimizaciones. El mismo realiza $N/2$ iteraciones (parte entera de la división), y en cada una de ellas ubica la reina en una posición diferente de la primera fila. Las $N/2$ posiciones restantes no son analizadas debido a que son soluciones simétricas de la rotación a 90^{o} de los casos evaluados.

Algoritmo 5.5 Pseudocódigo del bloque principal.

```
main () {
     cantSol:=0
     for (pos:1..N/2)
          ubicarReina (1,pos,tablero)
          detPosVálida (posVálida,tablero,2)
          cantSol:= cantSol + detSol (2,posVálida,tablero)
}
```

Algoritmo 5.6 Pseudocódigo del algoritmo que ubica las reinas en una posición válida.

```
function detSol (fila, posVálida, tablero) {
     i:= posiciónVálida (posVálida)
     if (fila = N) and (i ≤ N ) then
          ubicarReina (fila,i,tablero)
          return (cantidadSoluciones (tablero))
     else
          total:=0
          while (i ≤ N)
             ubicarReina (fila,i,tablero)
             detPosVálida (nuevaPosVálida,tablero, fila+1).
             total:= total + detSol (fila+1,nuevaPosVáalida,tablero )
             i:= posiciónVálida (posVálida)
          return total
     }
     //Ubicar una reina en la posición p de la fila f del tablero t.
     ubicarReina (f,p,t)
     //Determinar el conjunto c de posiciones válidas para la fila f en el tablero t.de 5 a 10
     detPosVálida (c,t,f)
     //Sacar y retornar la primer posición válida dada en c, o retornar N si no hay ninguna.
     posición Válida (c)
```

Algoritmo 5.7 Pseudocódigo del contador de soluciones derivadas.

```
function cantidadSoluciones (t) {
     if (rot(t,90)=t) or (rot(t,90)=sim(t)) then
          return (2)
     else
          if (rot(t,180)=t) or (rot(t,180)=sim(t)) then
             return (4)
          else return (8)
     }
     rot(t,g): retorna el tablero t rotado en g grados.
     sim(t): retorna el tablero simétrico de t.
}
```

En el pseudocódigo presentado en el Algoritmo 5.5 se observa que una vez ubicada la reina en la primera fila se determina el vector de posiciones válidas para la siguiente fila, y así para cada una de ellas en donde se calculan las soluciones que estas generan. Para obtener la cantidad de soluciones a partir de la fila i (tal que toda fila j con $j \leq i$ tiene ubicada su reina), se establece el vector de posiciones válidas para la fila $i + 1$, donde para cada una de ellas se vuelve a repetir este paso. Por tanto, esto se realiza hasta llegar a ubicar una reina en la última fila, o cuando no hay más posiciones válidas en una cierta fila (Figura 5.6). Finalmente, una vez ubicada la reina de la última fila se calcula la cantidad de soluciones diferentes que genera dicha combinación. Esto se logra, contabilizando la combinación original, su simétrica y rotaciones de la misma a $90^{\underline{o}}$, $180^{\underline{o}}$, $270^{\underline{o}}$ y sus respectivas simétricas (Figura 5.7).

Solución paralela

En esta solución, se ubica la reina de una o más filas y se obtienen todas las soluciones para esa disposición inicial. Cada uno de los procesos se encarga de resolver el problema para un subconjunto de éstas, de manera tal que el sistema completo trabaje con todas las posibles combinaciones (cada una será una tarea en la descomposición del problema) de esas filas. Para esto deben tenerse en cuenta dos aspectos, uno de ellos es la forma en la que se construyen las combinaciones y el otro aspecto es cómo distribuir las combinaciones entre las máquinas.

Para la construcción de las combinaciones, la idea inicial es que sólo se tenga en cuenta la primera fila del tablero ($N/2$ posibilidades como se expresa en la solución secuencial). Cada proceso se encarga de encontrar todas las soluciones a partir de la reina ubicada en una posición de dicha fila.

La cantidad de tableros a evaluar por el algoritmo se incrementa a medida que la reina ubicada en la primera fila se aproxima al centro del tablero [46]. Sin embargo, esta solución no es muy útil, visto que, son muy pocas tareas y al mismo tiempo muy irregulares.

A la hora de distribuir, lo más importante es balancear el trabajo, de manera que es conveniente usar solución de "grano fino". En otras palabras, muchas combinaciones de poco cómputo en donde cada una tiene el objetivo de poder nivelar el trabajo realizado por cada máquina resolviendo varias de estas [47]. Esto se puede lograr asignando más filas para formar cada una de las combinaciones a resolver; por ejemplo, utilizando las primeras i filas se obtienen $W = (N^i/2)$ combinaciones diferentes para disponer entre todos los núcleos, siendo N el tamaño del tablero. De hecho, cuanto más balanceado sea el trabajo a realizar entre los distintos procesadores, más se reduce el tiempo del algoritmo.

Estas combinaciones se distribuyen dinámicamente bajo demanda entre los procesos utilizando el paradigma de programación Master/Worker. Particularmente, esta estrategia requiere un proceso master que sólo se encarga de distribuir las combinaciones y recibir los resultados, es decir que no cumple el rol de worker. En esta solución la cantidad total de procesos es $p + 1$. Las W combinaciones son distribuidas por el master entre los $ntask - 1$ procesos workers, inicialmente distribuye un subconjunto de combinaciones equitativamente entre todos los workers, y luego, a medida que estos soliciten más trabajo se distribuyen bajo demanda las combinaciones restantes.

A continuación, en el Algoritmo 5.8 se presenta el pseudocódigo del proceso *master* que realiza esta distribución. En este pseudocódigo (como en las pruebas) se usan

cuatro filas del tablero para formar las combinaciones iniciales a distribuir.

En el Algoritmo 5.9 se muestra el pseudocódigo de la función detSolParcial que se utiliza para resolver una combinación inicial a partir de las reinas ubicadas en las primeras 4 filas (la función detSol es la presentada en el Algoritmo 5.8). En el Algoritmo 5.10 se muestra el pseudocódigo de los procesos Worker.

Algoritmo 5.8 Pseudocódigo del proceso master.

```
main () {

      cantSol := 0
      for (i:1..p) //Reparto inicial
            determina la primera combinación (pri)
            determina la última combinación (ult)
            send (pri, ult, i)
      while (hay combinaciones) //Reparto restante
            recv (pedido, x)
            determina la primera combinación (pri)
            determina la última combinación (ult)
            send (pri, ult, x)
      for (i:1..p)
            send (FIN, FIN, i)
      for (i:1..p) //Recepción de los resultados
            recv (cantOtroProc, i)
            cantSol := cantSol + cantOtroProc

}
```

Algoritmo 5.9 Pseudocódigo de la función detSolParcial.

```
function detSolParcial (posFila1, posFila2, posFila3, tablero) {

      ubicarReina (1, posFila1, tablero)
      ubicarReina (2, posFila2, tablero)
      ubicarReina (3, posFila3, tablero)
      detPosVálida (posVálida, tablero, 4)
      return detSol (4, posVálida, tablero)

}
//Ubicar una reina en la posición p de
//la fila f del tablero t.
ubicarReina (f,p,t)
//Determinar el conjunto c de posiciones
//válidas para la fila f en el tablero t.
detPosVálida (c,t,f)
```

5.2. Pruebas realizadas

Para realizar la experimentación se utilizó el cluster de RPi descrito en el Capítulo 4. Además, también se usó un cluster de computadoras x86, el cual cuenta con un procesador Intel Core i5-4460 3.20Ghz, 8 GB RAM y conexión de red Gigabit

Algoritmo 5.10 Pseudocódigo del proceso worker.

```
main () { //Proceso worker pid , id = 1..p

      cantSol := 0
      recv (pri, ult, 0)
      while (pri <> ult)
           for (combinacion:pri..ult)
                determina la ubicación en la fila 1 (q1)
                determina la ubicación en la fila 2 (q2)
                determina la ubicación en la fila 3 (q3)
                determina la ubicación en la fila 4 (q4)
                cantSol := cantSol + detSolParcial (q1,q2,q3,q4,tablero)
           send (pedido, 0)
           recv (pri, ult, 0)
      send (cantSol, 0)

}
```

Ethernet. Para el desarrollo de los algoritmos se usó el lenguaje C (gcc v5.4.0) junto a la librería OpenMPI (v3.0.0).

Las pruebas fueron realizadas en tres etapas:

- el objetivo de la primera etapa fue determinar los tiempos de ejecución de los algoritmos secuenciales, variando el tamaño de problema. Particularmente en el caso de la multiplicación de matrices, además de variar el tamaño del problema, se probaron diferentes tamaños de bloque a fin de obtener el óptimo, el cual se utilizó posteriormente para correr las pruebas paralelas.

- la segunda etapa de pruebas se basó en determinar los tiempos de ejecución de los algoritmos paralelos en el clúster de RPi. Es importante destacar que en las pruebas paralelas se decidió ejecutar las mismas con mapping manual, permitiendo asignar cada proceso a un determinado núcleo.

- en la tercera etapa se ejecutaron los algoritmos paralelos en el clúster de computadoras x86.

Para evaluar el comportamiento de los algoritmos desarrollados al escalar el problema y/o la arquitectura se analiza el speedup y la eficiencia, los cuales se describen en la Sección 2.7 del Capítulo 2.

Para mitigar posibles variaciones en los resultados de las pruebas, las mismas se ejecutaron 10 veces cada una, a fin de obtener la media de los datos obtenidos.

5.2.1. Primera etapa de pruebas

Para las 3 aplicaciones se varió el tamaño del problema. Para la multiplicación de matrices y para Jacobi se usó $N = \{1024, 2048, 4096, 8192\}$ mientras que para N-Reinas se empleó $N = \{16, 17, 18, 19\}$. En el caso de la multiplicación de matrices, también se varió el tamaño de bloque de forma de poder identificar el que resulta óptimo: $TB = \{32, 64, 128, 256, 512\}$.

5.2.2. Segunda etapa de pruebas

Para las pruebas de los algoritmos paralelos no sólo se varió el tamaño del problema sino también la cantidad de núcleos usados:

- los tamaños de problema fueron los mismos que en la primera etapa. En el caso particular de la multiplicación de matrices, se usó el tamaño de bloque identificado como óptimo en la etapa anterior[1].
- el número de núcleos empleados varió desde 4 hasta 64 ($P = 4, 8, 16, 32, 64$), lo que significó variar el número de nodos desde 1 hasta 16. Los procesos MPI fueron mapeados usando la opción hostfile, asignando de a 4 procesos por nodo.

5.2.3. Tercera etapa de pruebas

En esta etapa se ejecutan los algoritmos paralelos en el clúster de computadoras x86. Al igual que las pruebas realizadas en el cluster RPi, se varió tanto el tamaño de la arquitectura como el del problema.

5.3. Resultados obtenidos

En esta sección se muestran los resultados obtenidos en las etapas de pruebas mencionadas en la sección 5.2.

5.3.1. Primera etapa de pruebas

Resultados correspondientes a las pruebas secuenciales.

[1]El tamaño de bloque óptimo es utilizado por el algoritmo sólo si este es mayor al cociente entre el tamaño del conjunto de datos de entrada (tamaño de la matriz) y la cantidad de núcleos disponibles para resolver la solución del problema, por el contrario, se utiliza el resultado que se obtiene del cociente calculado. Es importante que esto sea así ya que existen casos en los que el bloque óptimo excede las dimensiones del tamaño del problema.

Multiplicación de matrices

En la Tabla 5.1 se puede observar los tiempos de ejecución de las distintas pruebas realizadas. A simple vista, se puede concluir que el tamaño de bloque óptimo es 128, dado que es el tamaño de bloque con el que se obtuvo menor tiempo, independientemente del tamaño del problema.

TB	N			
	1024	2048	4096	8192
32	14,79	107,98	876,38	7248,18
64	14,72	101,91	817,53	6569,66
128	14,39	98,62	793,83	6313,46
256	15,71	107,34	875,55	6877,46
512	15,03	100,66	794,00	6595,83

Tabla 5.1: Promedios de los tiempos de ejecución (en segundos) obtenidos con el algoritmo secuencial por bloques para los diferentes tamaños de problema (N) y de bloque TB utilizados.

Jacobi-solver

En la Tabla 5.2 se muestran los resultados obtenidos de la ejecución de la aplicación de prueba secuencial de Jacobi-solver.

P	N			
	1024	2048	4096	8192
1	103,87	486,90	2033,14	8088,26

Tabla 5.2: Promedios de los tiempos de ejecución (en segundos) obtenidos con el algoritmo secuencial para los diferentes tamaños de problema (N).

N-reinas

En la Tabla 5.3 se muestra los resultados obtenidos de la ejecución del algoritmo secuencial de N-reinas.

P	N			
	16	17	18	19
1	9,99	69,27	505,22	3887,27

Tabla 5.3: Promedios de los tiempos de ejecución (en segundos) obtenidos con el algoritmo secuencial para los diferentes tamaños de problema (N).

5.3.2. Segunda etapa de pruebas

Resultados correspondientes a las pruebas paralelas.

Multiplicación de matrices

La Tabla 5.4 muestra los tiempos de ejecución en segundos del algoritmo paralelo para multiplicación de matrices variando el tamaño del problema y la cantidad de procesadores usados. Se puede apreciar que no hay resultados para el tamaño de problema con $N = 8192$, lo cual se debe a que al algoritmo falla por excesivo consumo de memoria. Dado que, para alocar la matriz B es necesario contar con 256Mb porque $8192 \times 8192 \times 4\,bytes = \frac{268435456\,bytes}{1024 \times 1024} = 256$, si se tiene en cuenta de que es necesario tener una copia de esta matriz por proceso, en el peor caso se necesitaría disponer con al menos 1024Mb (256×4) para almacenar las matrices B en memoria. Además, hay que tener en cuenta que se necesita contar con memoria para alojar la porción de A asignada a cada uno de los procesos y la poción de C donde se irán almacenando los resultados de los cálculos. En definitiva, almacenar todas estas estructuras resulta imposible, dado que la cantidad de memoria disponible en un nodo RPi es de 918Mb, lo cual es significativamente menor a la cantidad necesaria para ejecutar la prueba.

P	N			
	1024	2048	4096	8192
4	4,97	33,53	793,83	-
8	3,00	19,84	135,83	-
16	2,19	12,98	79,01	-
32	1,78	9,91	47,68	-
64	1,96	7,66	44,00	-

Tabla 5.4: Promedios de los tiempos de ejecución (en segundos) obtenidos con el algoritmo paralelo por bloques para los diferentes tamaños de problema (N) y la cantidad de núcleos (P) utilizados.

La Tabla 5.5 presenta un resumen de los valores de los speedups obtenidos.

P	N			
	1024	2048	4096	8192
4	2,89	2,94	2,95	-
8	4,80	4,97	5,84	-
16	6,58	7,60	10,05	-
32	8,09	9,95	16,65	-
64	7,33	12,88	18,04	-

Tabla 5.5: Resumen de los speedups obtenidos con los algoritmo paralelos en las pruebas realizadas.

La Figura 5.6 ilustra los resultados de la Tabla 5.5.

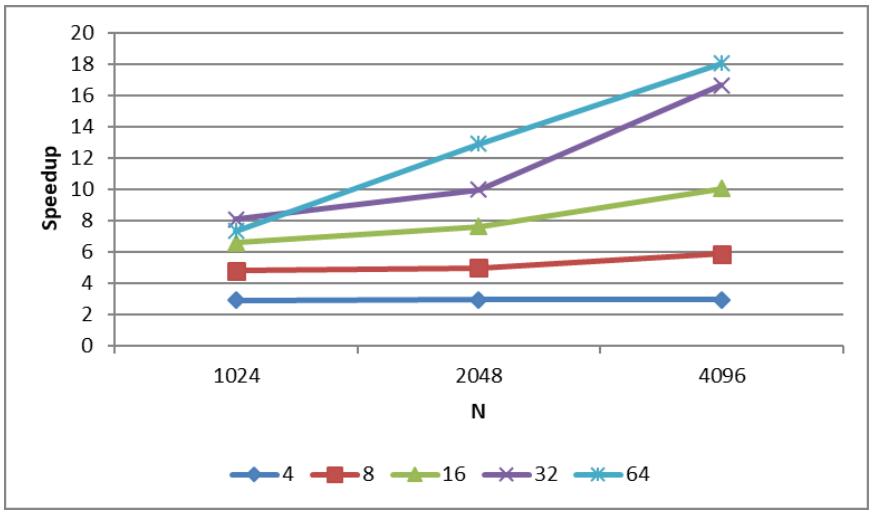


Figura 5.6: Speedups obtenidos por el algoritmo paralelo durante la fase de pruebas.

En la Figura 5.6 se puede apreciar como el speedup crece tanto al aumentar el número núcleos empleados como el tamaño de problema utilizado.

En la Tabla 5.6 se muestran los resultados de la eficiencia obtenida para el algoritmo paralelo MPI. Asimismo, en un sistema paralelo usualmente se espera que se cumpla: (1) que la eficiencia se incremente a medida que crece el tamaño del problema dejando fijo el número de núcleos usados (2) que la eficiencia se degrade al incrementar el numero de núcleos y dejar fijo el tamaño del problema; se puede observar claramente en el gráfico de la Figura 5.7 que estás dos situaciones se cumplen.

Este decremento de eficiencia se debe principalmente al overhead que generan las comunicaciones. A medida que aumenta el tamaño del problema es necesario trans-

mitir más cantidad de datos, y si también se tiene en cuenta el incremento de núcleos, aumentan las comunicaciones entre procesos. Es claro que este comportamiento impacta directamente sobre la eficiencia del algoritmo.

P	N			
	1024	2048	4096	8192
4	0,72	0,74	0,74	-
8	0,60	0,62	0,73	-
16	0,41	0,48	0,63	-
32	0,25	0,31	0,52	-
64	0,11	0,20	0,28	-

Tabla 5.6: Eficiencias obtenidas con el algoritmo paralelo en las pruebas realizadas en la segunda fase.

El máximo speedup alcanzado es 18,04 y se logra con $N = 4096$ y $P = 64$, aunque la eficiencia asociada es baja (0,28). Por otra parte, la máxima eficiencia se logra cuando $N = 4096$ y $P = 4$. Esto se debe principalmente a que las comunicaciones son realizadas dentro del mismo nodo y no hay necesidad de acceder a la red para realizar comunicaciones a otros nodos del clúster. Al utilizar más núcleos, la eficiencia disminuye debido al aumento del overhead (se produce un incremento de la cantidad de comunicaciones).

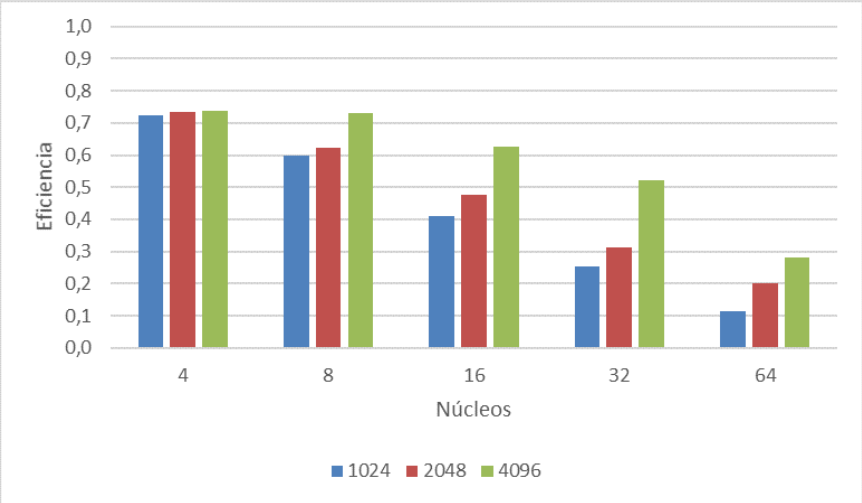


Figura 5.7: Eficiencias obtenidas con algoritmo paralelo para diferentes tamaños de problema y cantidad de núcleos.

Como se vio anteriormente, el overhead generado por la interacción entre procesos limita la eficiencia de las pruebas realizadas. Resulta interesante analizar la incidencia de las comunicaciones en el rendimiento final. A continuación en la Tabla 5.7 se muestran los tiempos totales de comunicación calculados para cada conjunto de pruebas paralelas realizadas.

P	N			
	1024	2048	4096	8192
4	0,03	0,41	0,83	-
8	0,77	3,70	14,62	-
16	0,97	4,66	18,60	-
32	1,37	6,32	24,80	-
64	1,74	6,03	29,25	-

Tabla 5.7: Promedios de los tiempos de comunicaciones (en segundos) del algoritmo paralelo para los diferentes tamaños de problema (N) y cantidad de núcleos (P)

En la Tabla 5.8 pueden apreciarse los porcentajes de comunicación involucrado en cada una de las pruebas. El porcentaje de comunicación se calculó de la siguiente forma: $(\frac{Tiempodecomunicaciones}{Tiempototal}) * 100$

P	N			
	1024	2048	4096	8192
4	0,69	1,23	0,31	-
8	25,76	18,63	10,76	-
16	44,31	35,91	23,55	-
32	76,85	63,78	52,01	-
64	88,53	78,77	66,47	-

Tabla 5.8: Porcentajes de comunicaciones obtenidos con los algoritmos paralelos para los diferentes tamaños de problema (N) y cantidad de núcleos (P) utilizados.

El gráfico de la Figura 5.8 plasma los porcentajes de comunicación de la tabla anterior.

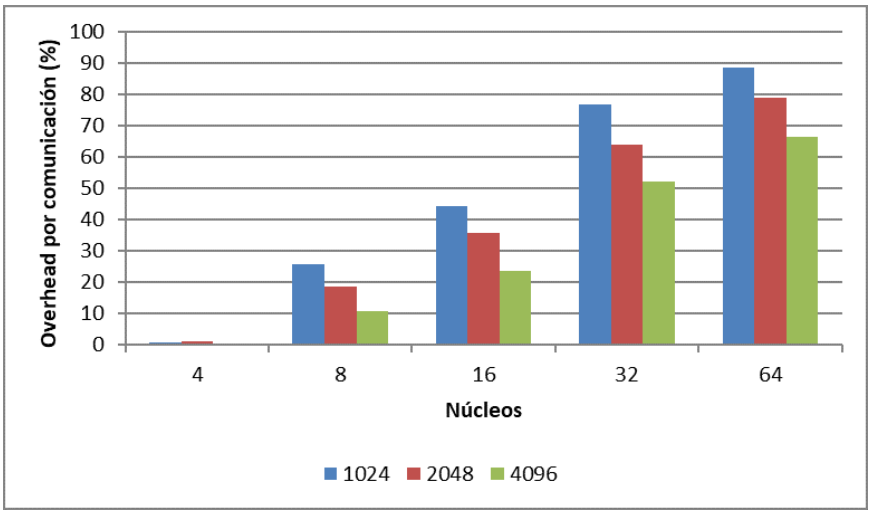


Figura 5.8: Porcentaje de overhead por comunicación obtenidos por los algoritmos paralelos para diferentes tamaños de problema y cantidad de núcleos.

Se puede observar el efecto que produce aumentar la cantidad de núcleos. Al aumentar la misma y mantener el tamaño del problema, el porcentaje de comunicación aumenta. A la inversa, el porcentaje de comunicación disminuye cuando el tamaño de problema aumenta y el numero de núcleos de mantiene. También se observan porcentajes elevados de overhead que obviamente penalizan el rendimiento. Esto se debe a la velocidad de la interfaz de red de la RPi (100Mbit), lo cual impone una fuerte limitación al rendimiento final de la aplicación considerando el volumen de las comunicaciones. La excepción es cuando $P = 4$, ya que no se genera comunicación real por la red.

Jacobi-solver

En la Tabla 5.9 se muestran los valores obtenidos de las ejecuciones de las pruebas paralelas.

P	N			
	1024	2048	4096	8192
4	45,28	196,32	799,95	3362,19
8	37,70	119,71	424,38	1686,95
16	26,47	75,01	240,33	898,09
32	19,73	53,50	144,87	487,15
64	21,88	50,88	144,57	466,30

Tabla 5.9: Promedios de los tiempos de ejecución (en segundos) obtenidos con el algoritmo paralelo para los diferentes tamaños de problema (N) y la cantidad de núcleos (P) utilizados.

En la Tabla 5.10 se visualizan los speedups calculados para cada una de las pruebas realizadas.

P	N			
	1024	2048	4096	8192
4	2,29	2,48	2,54	2,41
8	2,75	4,07	4,79	4,79
16	3,92	6,49	8,46	9,01
32	5,26	9,10	14,03	16,60
64	4,75	9,57	14,06	17,35

Tabla 5.10: Resumen de los speedups obtenidos con el algoritmo paralelo en las pruebas realizadas.

La Figura 5.9 muestra la tendencia de los mismos.

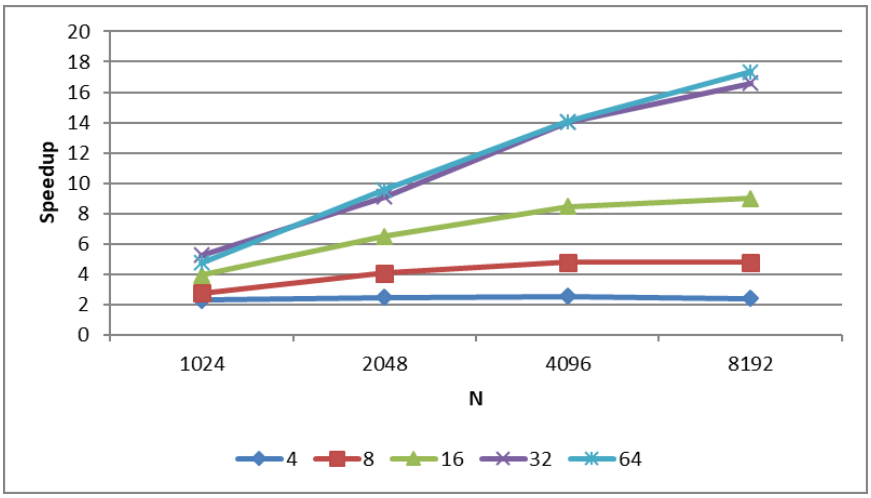


Figura 5.9: Speedups obtenidos en el algoritmo paralelo durante la fase de pruebas.

En la Tabla 5.11 se presentan los valores de las eficiencias resultantes del algoritmo paralelo. En la Figura 5.10 se grafican los valores de esta tabla para facilitar su comprensión.

P	N			
	1024	2048	4096	8192
4	0,57	0,62	0,64	0,60
8	0,34	0,51	0,60	0,60
16	0,25	0,41	0,53	0,56
32	0,16	0,28	0,44	0,52
64	0,07	0,15	0,22	0,27

Tabla 5.11: Eficiencias obtenidas con el algoritmo paralelo en las pruebas realizadas en la segunda fase.

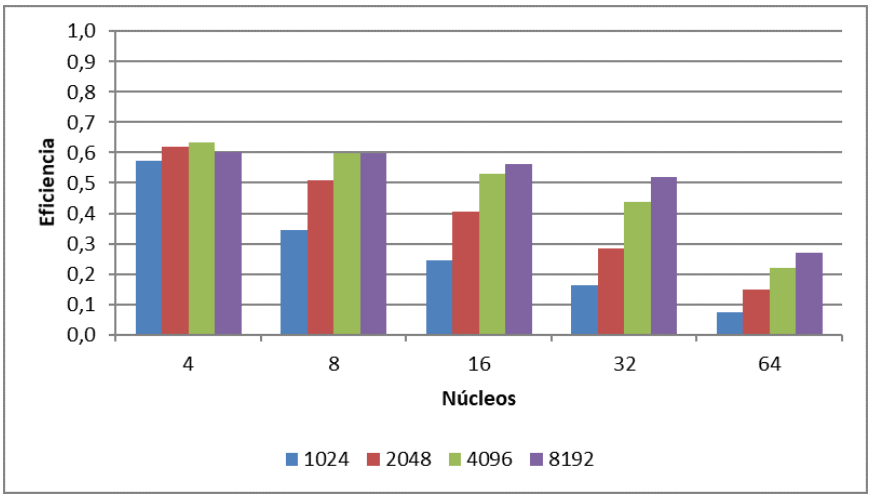


Figura 5.10: Eficiencias obtenidas con el algoritmo paralelos para diferentes tamaños de problema y cantidad de núcleos.

De los resultados se observa que al mantener fijo el tamaño de la matriz y aumentar la cantidad de procesadores, el speedup obtenido mejora; sin embargo, aunque se resuelve el problema en menos tiempo, la eficiencia disminuye.

La disminución de la eficiencia se debe al overhead de comunicación que se genera al distribuir las matrices y al comunicar los bordes de las mismas a cada proceso. Al escalar el problema manteniendo fijo el número de núcleos, tal como se ve en la Tabla 5.10 y la Figura 5.10, tanto el speedup como la eficiencia mejoran en general, dado que el overhead mencionado anteriormente tiene menor peso en el tiempo total de procesamiento.

Se debe notar que los valores de eficiencia obtenidos con la prueba de Jacobi-solver son inferiores a los valores de eficiencia de la prueba de Multiplicación de matrices. Esto es debido a que, aunque manejen estructuras de datos más pequeñas, deben realizarse más comunicaciones (de pocos datos) y la relación cómputo-comunicación es mucho mas baja. Por otro lado, también es importante mencionar que existen muchos puntos de sincronización que también producen overhead (espera ociosa).

N-reinas

La Tabla 5.12 muestra los tiempos de ejecución obtenidos para las diferentes pruebas realizadas.

P	N			
	16	17	18	19
4	3,49	24,19	175,98	1348,38
8	1,58	10,43	75,87	577,72
16	1,30	5,95	35,77	269,75
32	0,49	2,49	17,97	130,70
64	0,47	1,99	9,83	75,20

Tabla 5.12: Promedios de los tiempos de ejecución (en segundos) obtenidos con el algoritmo paralelo para los diferentes tamaños de problema (N) y la cantidad de núcleos (P) utilizados.

Por otro lado, la Tabla 5.13 muestra los speedups alcanzados por el algoritmo paralelo, mientras que la Figura 5.11 los ilustra. Como es de esperar el speedup aumenta al incrementar la arquitectura sobre la que se realiza la prueba. Por otro lado, al aumentar el tamaño del tablero en una misma arquitectura, se observa un incremento del speedup, tendiendo a estabilizarse a partir de N=18.

P	N			
	16	17	18	19
4	2,86	2,86	2,87	2,88
8	6,33	6,64	6,66	6,73
16	7,70	11,63	14,13	14,41
32	20,51	27,80	28,12	29,74
64	21,38	34,83	51,39	51,69

Tabla 5.13: Resumen de los speedups obtenidos con el algoritmo paralelo.

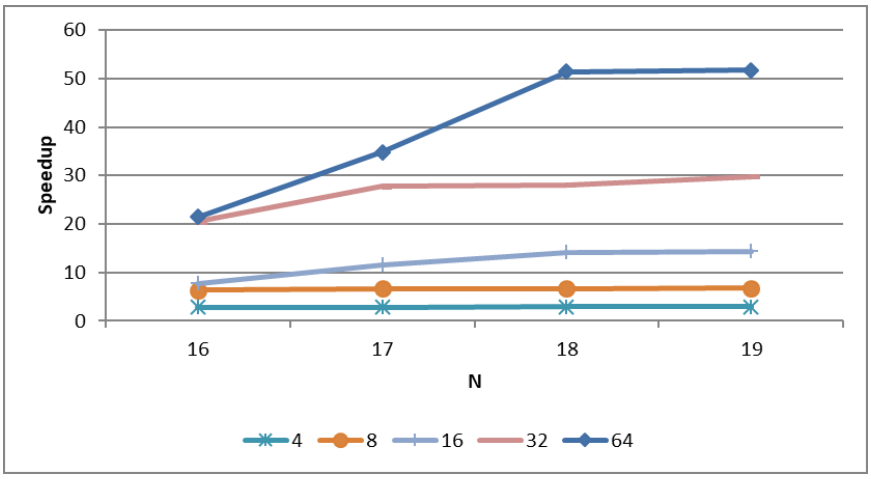


Figura 5.11: Speedups obtenidos por los algoritmos paralelos durante la fase de pruebas.

En la Tabla 5.14 se visualizan las eficiencias logradas mientras que la Figura 5.12 los gráfica.

P	N			
	16	17	18	19
4	0,72	0,72	0,72	0,72
8	0,79	0,83	0,83	0,84
16	0,48	0,73	0,88	0,90
32	0,64	0,87	0,88	0,93
64	0,33	0,54	0,80	0,81

Tabla 5.14: Eficiencias obtenidas con el algoritmo paralelo en las pruebas realizadas en la segunda fase.

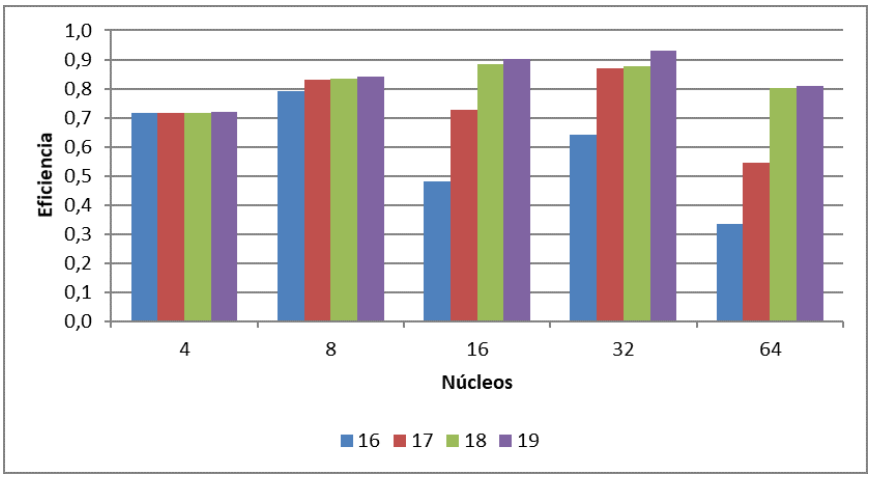


Figura 5.12: Eficiencias obtenidas por el algoritmos paralelos para diferentes tamaños de problema y cantidad de núcleos.

Desde el punto de vista de la eficiencia se observa un incremento a medida que se aumenta el tamaño del problema. Esto se debe a que el componente paralelo del algoritmo crece a un ritmo mayor que el componente serie al aumentar el problema (tamaño del tablero). Esto suele caracterizar a los problemas escalables. Por otro lado, también se logra un incremento en la eficiencia -más suave- al incrementar el número de núcleos hasta 32, luego (con 64) se reduce.

Cabe recalcar que de las pruebas realizadas, la prueba de N-reinas fue la que mejores resultados arrojó, por sus valores de eficiencia cercanos a 1. Esto se debe a diferentes causas. En primer lugar, la aplicación trabaja con un conjunto reducido de datos de tamaño mucho menor al del resto de los benchmarks seleccionados. Adicionalmente, si bien requiere una gran cantidad de comunicaciones, las mismas son de pocos datos y no representan un porcentaje de tiempo significativo respecto al tiempo total de la aplicación. Por último, el algoritmo trabaja con número enteros a diferencia de los anteriores que computan en punto flotante.

5.4. Comparación con cluster de multicores x86

En esta sección se procede a realizar la comparación con el clúster de multicores, no sólo desde el punto de vista del rendimiento, sino también de la eficiencia energética.

5.4.1. Tiempo de ejecución

A continuación se visualizan los resultados obtenidos en la tercer etapa de pruebas.

Multiplicación de matrices

En la Tabla 5.15 se muestran los tiempos promedios obtenidos de las distintas ejecuciones de la prueba de Multiplicación de matrices ejecutada en el clúster de computadoras.

P	N			
	1024	2048	4096	8192
4	0,27	2,15	17,21	190,60
8	0,30	1,42	9,91	99,77
16	0,30	1,25	7,88	72,14
32	0,28	1,52	5,83	40,54
64	0,43	1,33	4,21	24,55

Tabla 5.15: Promedios de los tiempos de ejecución (en segundos) obtenidos en el cluster de multicores con el algoritmo paralelo para los diferentes tamaños de problema (N) y la cantidad de núcleos (P) utilizados.

Jacobi solver

En la Tabla 5.16 se muestran los tiempos promedios obtenidos de las distintas ejecuciones de la prueba de Jacobi-solver ejecutada en el clúster de computadoras.

P	N			
	1024	2048	4096	8192
4	4,87	35,67	139,74	559,37
8	4,88	17,94	70,51	280,17
16	5,08	6,71	22,56	90,11
32	6,26	4,95	11,88	44,77
64	5,85	4,81	7,15	24,77

Tabla 5.16: Promedios de los tiempos de ejecución (en segundos) obtenidos en el cluster de multicores con el algoritmo paralelo para los diferentes tamaños de problema (N) y la cantidad de núcleos (P) utilizados.

N-reinas

En la Tabla 5.17 se muestran los tiempos promedios obtenidos de las distintas ejecuciones de la prueba de *N-reinas* ejecutada en el clúster de computadoras.

P	N			
	16	17	18	19
4	0,74	5,18	37,55	287,73
8	0,36	2,26	16,13	123,69
16	0,27	1,32	9,17	70,11
32	0,24	0,75	4,58	34,25
64	0,44	0,62	2,46	16,94

Tabla 5.17: Promedios de los tiempos de ejecución (en segundos) obtenidos en el cluster de multicores con el algoritmo paralelo para los diferentes tamaños de problema (N) y la cantidad de núcleos (P) utilizados.

5.4.2. Rendimiento

Para poder comparar el rendimiento entre los diferentes clústers se requiere una métrica común. Es por eso que se seleccionó la métrica FLOPS para la multiplicación de matrices y la iteración de Jacobi. Sin embargo, en el caso de N-Reinas, al ser una aplicación que computa con números enteros, se empleó una métrica ad-hoc. Esta métrica considera el número total de reinas posicionadas y se la denominó MQUEENS (millones de reinas posicionadas).

Multiplicación de matrices

P	N			
	1024	2048	4096	8192
4	0,43	0,51	0,51	-
8	0,72	0,87	1,01	-
16	0,98	1,32	1,74	-
32	1,21	1,73	2,88	-
64	1,09	2,24	3,12	-

(a) Cluster RPi 3

P	N			
	1024	2048	4096	8192
4	7,85	7,99	7,99	-
8	7,19	12,09	13,86	-
16	7,17	13,73	17,43	-
32	7,78	11,32	23,58	-
64	5,02	12,95	32,62	-

(b) Cluster i5-4460

Tabla 5.18: GFLOPS obtenidos para los diferentes tamaños de problema (N) y cantidad de núcleos (P) utilizados en ambos clusters.

Jacobi-solver

P	N			
	1024	2048	4096	8192
4	0,74	0,68	0,67	0,64
8	0,89	1,12	1,27	1,27
16	1,27	1,79	2,23	2,39
32	1,70	2,51	3,71	4,41
64	1,53	2,64	3,71	4,61

(a) Cluster RPi 3

P	N			
	1024	2048	4096	8192
4	6,88	3,76	3,84	3,84
8	6,88	7,48	7,61	7,67
16	6,61	20,01	23,79	23,83
32	5,36	27,13	45,18	47,96
64	5,73	27,92	75,07	86,71

(b) Cluster i5-4460

Tabla 5.19: GFLOPS obtenidos para los diferentes tamaños de problema (N) y cantidad de núcleos (P) utilizados en ambos clusters.

N-reinas

P	N			
	16	17	18	19
4	69,95	70,06	70,22	70,66
8	154,63	162,51	162,88	164,92
16	188,12	284,56	345,50	353,21
32	501,28	680,00	687,74	728,97
64	522,59	852,10	1257,04	1266,97

(a) Cluster RPi 3

P	N			
	16	17	18	19
4	327,88	327,27	329,12	331,14
8	679,39	750,89	766,08	770,29
16	888,43	1284,37	1347,85	1358,97
32	1035,69	2259,81	2697,68	2782,20
64	554,17	2753,42	5014,06	5623,87

(b) Cluster i5-4460

Tabla 5.20: MQUEENS obtenidos para los diferentes tamaños de problema (N) y cantidad de núcleos (P) utilizados en ambos clusters..

Como se puede apreciar en las tablas 5.18, 5.19 y 5.20, en ambos cluster y para todas las aplicaciones el rendimiento global del sistema mejora al aumentar tanto el tamaño del problema como de la arquitectura. Como era de esperar, el cluster x86 obtiene rendimientos superiores en las 3 aplicaciones. Esto se debe a que sus procesadores son más potentes y sofisticados, sus memorias RAM son más rápidas, su interfaz de red es más veloz, entre otras características más avanzadas de este cluster.

Por otra parte, a nivel de aplicación, la mayor diferencia[2] se encuentra en la multiplicación de matrices con una diferencia promedio de 10.4× y una máxima de 18.2×, seguido por Jacobi con promedio de 9× y máxima de 18.8×; y por último, N-Reinas donde la diferencia es mucho menor, promedio de 4× y máxima de 4.7×.

[2]La diferencia se calcula como: flops_i5/flops_rpi3

5.4.3. Eficiencia energética

En sentido similar a la sección anterior, para poder evaluar la eficiencia energética se empleó la métrica FLOPS/Watt para la multiplicación de matrices y la iteración de Jacobi mientras que MQUEENS/Watt para N-Reinas. Para estimar los Watt consumidos por el cluster de procesadores Core i5, se empleó el valor observado en diferentes estudios realizados [48, 49], el cual a su vez coincide con el TDP[3] de los procesadores (84W). Por otro lado, en el caso del cluster de RPi, se optó por emplear el valor observado como máximo en diferentes reportes disponibles en la literatura (3.7W)[4] [51, 52, 53].

Multiplicación de matrices

P	N			
	1024	2048	4096	8192
4	0,12	0,14	0,14	-
8	0,10	0,12	0,14	-
16	0,07	0,09	0,12	-
32	0,04	0,06	0,10	-
64	0,02	0,04	0,05	-

(a) Cluster RPi 3

P	N			
	1024	2048	4096	8192
4	0,09	0,10	0,10	-
8	0,04	0,07	0,08	-
16	0,02	0,04	0,05	-
32	0,01	0,02	0,04	-
64	0,00	0,01	0,02	-

(b) Cluster i5-4460

Tabla 5.21: GFLOPS /W obtenidos para los diferentes tamaños de problema (N) y cantidad de núcleos (P) utilizados en ambos clusters.

[3]Aunque este valor no define la máxima cantidad de potencia que se podría consumir, es usualmente aceptado como un valor representativo del consumo de potencia para una gran cantidad de cargas de trabajo [50].

[4]En este caso el fabricante no ha especificado el TDP.

Jacobi-solver

P	N			
	1024	2048	4096	8192
4	0,20	0,18	0,18	0,17
8	0,12	0,15	0,17	0,17
16	0,09	0,12	0,15	0,16
32	0,06	0,08	0,13	0,15
64	0,03	0,04	0,06	0,08

(a) Cluster RPi 3

P	N			
	1024	2048	4096	8192
4	0,08	0,04	0,05	0,05
8	0,04	0,04	0,05	0,05
16	0,02	0,06	0,07	0,07
32	0,01	0,04	0,07	0,07
64	0,00	0,02	0,06	0,06

(b) Cluster i5-4460

Tabla 5.22: GFLOPS/W obtenidos para los diferentes tamaños de problema (N) y cantidad de núcleos (P) utilizados en ambos clusters.

N-reinas

P	N			
	16	17	18	19
4	18,90	18,94	18,98	19,10
8	20,90	21,96	22,01	22,29
16	12,71	19,23	23,34	23,87
32	16,94	22,97	23,23	24,63
64	8,83	14,39	21,23	21,40

(a) Cluster RPi 3

P	N			
	16	17	18	19
4	3,90	3,90	3,92	3,94
8	4,04	4,47	4,56	4,59
16	2,64	3,82	4,01	4,04
32	1,54	3,36	4,01	4,14
64	0,41	2,05	3,73	4,18

(b) Cluster i5-4460

Tabla 5.23: MQUEENS/W obtenidos para los diferentes tamaños de problema (N) y cantidad de núcleos (P) utilizados en ambos clusters.

A diferencia del análisis de rendimiento, es el cluster de RPi quien obtiene los mejores cocientes de eficiencia energética para todos las combinaciones (N,P). Esto se debe en gran medida a su bajo consumo de potencia. En particular, obtiene una diferencia promedio de 2.5× y una máxima de 5×, 3.1× con una máxima de 7.2× y 6.5× con máxima de 21.4×, para la multiplicación de matrices, Jacobi y N-Reinas, respectivamente. Es claro que la mayor diferencia se da en N-Reinas, ya que justamente es la aplicación en la que el cluster de RPi obtiene el mejor rendimiento y la menor diferencia con el cluster de i5.

Es importante aclarar que en el caso del cluster de RPi se está considerando el consumo completo de la placa, mientras que en el de i5, sólo el del procesador. Como otros componentes del sistema pueden tener consumos significativos[54], las diferencias a favor del cluster de RPi podrían ser aun mayores.

5.5. Trabajos similares

Existe un gran número de trabajos que describen cómo desplegar un cluster de RPi. Se hará foco en aquellos que realizan evaluaciones de rendimiento y/o eficiencia energética, además de comparaciones con otras arquitecturas.

Kiepert [55] describe el armado de un clúster de 32 RPi's. Para evaluar el rendimiento del mismo, ejecutó una aplicación MPI que estima el valor del número Pi. Al ser una aplicación embarazosamente paralela [9], el cluster obtiene aceleraciones muy cercanas al número de nodos empleados.

Die [56] desplegó un cluster de 8 RPi's, ejecutando OpenFOAM [57] y High-Performance Linpack (HPL) [58] para el análisis de rendimiento. El cluster armado mostró buenos rendimiento para OpenFOAM en los casos en que el volumen de comunicaciones era bastante menor que el de cómputo. Respecto a HPL, Die encontró niveles de prestaciones aceptables mientras los requerimientos de memoria no superaron la memoria RAM disponible. En esta tesina se observaron resultados similares, aunque al contar con más memoria RAM en cada placa, fue posible procesar problemas de mayor tamaño.

Cox et al [59] armaron un cluster de 64 RPi modelo B (usando uno de ellas como frontend). Al igual que el caso anterior, emplearon HPL para evaluar el rendimiento alcanzando 1.14 GFLOPS con el sistema completo. En forma similar a este trabajo, observaron que la capacidad de memoria de las placas, impiden computar problemas más grandes.

Un cluster de 25 RPi modelo B fue construido por Pfalzgraf y Driscoll [60]. Usando 3 aplicaciones de álgebra lineal (triada de vector, multiplicación de matriz-vector y multiplicación de matrices) analizaron el rendimiento del cluster armado. En el caso de la multiplicación de matrices, logran un pico de 1.1 GFLOPS con 24 placas. En esta tesina, se alcanza un pico de 3.12 GFLOPS con 16 placas, aunque obviamente se debe tener en cuenta la mayor potencia de las RPi 3 comparado a la Pi modelo B.

Cloutier et al [61] armaron un cluster de 25 RPi 2 modelo B. Una vez más, HPL fue empleado para el análisis de rendimiento. En este caso además, evaluaron la eficiencia energética del cluster midiendo el consumo a través de un medidor WattsUp

Pro. En la comparación con un sistema basado en un Intel Haswell EP de 16 núcleos, observan que el cluster de RPi se ve superado por el sistema x86 en forma opuesta a lo observado al momento. En su trabajo, los autores indican que entre las principales causas se encuentra el profundo nivel de optimización de HPL como benchmark para medir rendimiento y la alta tasa de operaciones en punto flotante, lo que favorece a los procesadores x86 tradicionales.

Cicirello [62] construyó un cluster de RPi 3 modelo B, seleccionando como casos de estudio la estimación del número Pi y la multiplicación de matrices. Lamentablemente, la ausencia de tiempos de ejecución o FLOPS impide la comparación directa con los resultados de esta tesina. Más allá de eso, y en sentido similar a este trabajo, Cicirello detecta que el rendimiento del cluster está significativamente limitado por la velocidad de la interfaz de red en aquellas aplicaciones que tienen comunicaciones de gran volumen. Además, encuentra aceptables tasas de aceleraciones en aplicaciones donde la cantidad y el volumen de comunicaciones tienen mucho menor peso que la proporción de cómputo.

Por último, los mismos autores de [61] desplegaron un cluster de RPi 3 B en [63]. Como era de esperar, el rendimiento de este cluster resulta superior al de[61] por usar placas más avanzadas. De todas formas, el uso de HPL como caso de estudio los lleva a encontrar que los sistemas x86 resultan superiores no sólo en rendimiento pico sino también en rendimiento/watt.

5.6. Resumen

En este capítulo se detallan y describen el conjunto de pruebas que se utilizaron para evaluar el rendimiento del clúster presentado en el Capítulo 4. Para ello se seleccionaron 3 aplicaciones que se pueden considerar clásicas y representativas de aplicaciones científicas, que poseen alta demanda computacional pero con diferentes características entre sí.

Las pruebas elegidas son: Multiplicación de matrices, por ser una aplicación masivamente paralela que demanda gran cantidad de cómputo regular, que requiere poca cantidad de comunicaciones al inicio y al final de cada tarea, pero de gran tamaño. Jacobi-solver, por ser una aplicación regular con alto grado de paralelismo y dependencia a nivel de datos entre las distintas tareas y etapas de ejecución. Esta gran cantidad de sincronización entre las tareas genera una considerable cantidad de comunicaciones de moderado tamaño a lo largo de toda la ejecución de la aplicación. Y por ultimo, N-reinas que fue elegida por ser una aplicación con alto grado de para-

lelismo, pero las tareas son irregulares y de gran tamaño, lo que obliga a realizar la distribución dinámica de esas tareas entre los procesos. Esto genera una gran cantidad de comunicaciones no estructuradas de tamaño pequeño.

Para realizar la experimentación se utilizó el cluster de RPi, además, de un cluster de computadoras x86, el cual cuenta con un procesador Intel Core i5-4460 3.20Ghz, 8 GB RAM y conexión de red Gigabit Ethernet. Las pruebas fueron realizadas en tres etapas:

- el objetivo de la primera etapa fue determinar los tiempos de ejecución de los algoritmos secuenciales, variando el tamaño de problema.
- la segunda etapa de pruebas se basó en determinar los tiempos de ejecución de los algoritmos paralelos en el clúster de RPi.
- en la tercera etapa se ejecutaron los algoritmos paralelos en el clúster de computadoras x86.

De los resultados obtenidos en la primera y segunda etapa de pruebas, se puede concluir que:

- aquellas aplicaciones que demandan comunicaciones de gran volumen, probablemente sufran una degradación importante en su rendimiento debido a la limitada velocidad de la interfaz de red que tiene la RPi, como se observó en la multiplicación de matrices.
- aquellas aplicaciones que requieran una gran cantidad de comunicaciones de tamaño moderado, también penalizarán su rendimiento por la misma causa que el caso anterior. Este comportamiento se observó con Jacobi-Solver.
- aplicaciones que trabajen con volúmenes de datos muy superiores a la capacidad disponible de la memoria RAM de las RPi, pueden sufrir errores en ejecución, tal como se vio en las pruebas de la multiplicación de matrices.
- aplicaciones que demanden mucho cómputo pero con pequeños datos, que no tengan una incidencia significativa de comunicaciones, probablemente tengan muy buen rendimiento. Esto se ve reflejado en el caso de N-Reinas, donde sus resultados muestran niveles altos de eficiencia.

Desde el punto de vista del rendimiento en ambos cluster y para todas las aplicaciones el rendimiento global del sistema mejora al aumentar tanto el tamaño del problema

como de la arquitectura. Como era de esperar, el cluster x86 obtiene rendimientos superiores en las 3 aplicaciones. Esto se debe a que sus procesadores son más potentes y sofisticados, sus memorias RAM son más rápidas, su interfaz de red es más veloz, entre otras características más avanzadas de este cluster.

Por otra parte, a nivel de aplicación, la mayor diferencia se encuentra en la multiplicación de matrices con una diferencia promedio de 10.4× y una máxima de 18.2×; seguido por Jacobi con promedio de 9× y máxima de 18.8×; y por último, N-Reinas donde la diferencia es mucho menor, promedio de 4× y máxima de 4.7×.

Para poder evaluar la eficiencia energética se empleó la métrica FLOPS/Watt para la multiplicación de matrices y la iteración de Jacobi mientras que MQUEENS/Watt para N-Reinas. A diferencia del análisis de rendimiento, es el cluster de RPi quien obtiene los mejores cocientes de eficiencia energética para todos las combinaciones (N,P). Esto se debe en gran medida a su bajo consumo de potencia. En particular, obtiene una diferencia promedio de 2.5× y una máxima de 5×, 3.1× con una máxima de 7.2× y 6.5× con máxima de 21.4×, para la multiplicación de matrices, Jacobi y N-Reinas, respectivamente. Es claro que la mayor diferencia se da en N-Reinas, ya que justamente es la aplicación en la que el cluster de RPi obtiene el mejor rendimiento y la menor diferencia con el cluster de i5.

Por último, del análisis de trabajos relacionados realizado, se puede decir que otros estudios también han identificado a la capacidad de memoria y la interfaz de red de las RPi como elementos que limitan el rendimiento alcanzable por las mismas. También que, al igual que en esta tesina, las RPi obtienen muy buenos resultados de aceleración en aplicaciones con pocas comunicaciones.

Capítulo 6

Conclusiones y trabajos futuros

En este capítulo se presentan las conclusiones obtenidas del desarrollo de esta tesina y los posibles trabajos futuros que se desprenden de la misma.

6.1. Conclusiones

Durante décadas, el desarrollo de plataformas HPC estuvo focalizado casi únicamente en mejorar el rendimiento de las mismas. Esto provocó un crecimiento exponencial en los requerimientos de potencia de estos sistemas (no sólo para alimentarlos sino también para refrigerarlos), lo que a su vez repercutió en el costo económico de los mismos. Es por eso que, hoy en día, la reducción del consumo energético se ha vuelto unos de los principales desafíos para la comunidad HPC.

La Raspberry Pi (RPi) es una SBC (por sus siglas en inglés: Single Board Computer) de bajo costo la cual cuenta con múltiples núcleos, pueden conectarse en red y dan soporte a sistemas operativos y herramientas de programación tradicionales de HPC. Aunque sus núcleos no son potentes, el bajo consumo de potencia de las RPi las vuelven una opción atractiva desde el punto de vista energético. Por esta razón, el objetivo de esta tesina se centró en armar un cluster de placas RPi y analizar la viabilidad de su uso para HPC.

El cluster desplegado consta de 20 placas RPi3 Modelo B (1 nodo master y 19 nodos worker), alimentadas por una fuente de PC e interconectadas entre sí por medio de un switch ethernet. Se decidió instalar Raspbian como sistema operativo por estar basado el Linux Debian Stretch y por la compatibilidad que tienen con él las herramientas elegidas para administrar el cluster. Los principales servicios que se configuraron en el nodo master son: NFS, NAT, DHCP, SSH y MPI. Y en el caso de

los nodos worker, que solo ejecutan servicios como: SSH y NFS.

Para poder evaluar el rendimiento y la eficiencia energética del cluster se seleccionaron 3 aplicaciones que se pueden considerar clásicas y representativas de aplicaciones científicas, que poseen alta demanda computacional. Estas aplicaciones son: la multiplicación de matrices, Jacobi-solver y el problema de las N-reinas.

En el análisis realizado sobre los resultados obtenidos de las diferentes pruebas se concluyó que:

- aquellas aplicaciones que demandan comunicaciones de gran volumen, probablemente sufran una degradación importante en su rendimiento debido a la limitada velocidad de la interfaz de red que tiene la RPi, como se observó en la multiplicación de matrices.

- aquellas aplicaciones que requieran una gran cantidad de comunicaciones de tamaño moderado, también penalizarán su rendimiento por la misma causa que el caso anterior. Este comportamiento se observó con Jacobi-Solver.

- aplicaciones que trabajen con volúmenes de datos muy superiores a la capacidad disponible de la memoria RAM de las RPi, pueden sufrir errores en ejecución, tal como se vio en las pruebas de la multiplicación de matrices.

- aplicaciones que demanden mucho cómputo pero con pequeños datos, que no tengan una incidencia significativa de comunicaciones, probablemente tengan muy buen rendimiento. Esto se ve reflejado en el caso de N-Reinas, donde sus resultados muestran niveles altos de eficiencia.

Como era de esperar, desde el punto de vista del rendimiento, el cluster desplegado en esta tesina presenta tasas de prestaciones más bajas en comparación a un cluster de procesadores x86. En la experimentación se observó que, a nivel de aplicación, la mayor diferencia se encuentra en la multiplicación de matrices con una diferencia promedio de 10.4× y una máxima de 18.2×; seguido por Jacobi con promedio de 9× y máxima de 18.8×; y por último, N-Reinas donde la diferencia es mucho menor, promedio de 4× y máxima de 4.7×. Sin embargo, como desde el punto de vista de eficiencia energética el cluster de RPi obtiene una diferencia promedio de 2.5× y una máxima de 5×, 3.1× con una máxima de7.2× y 6.5× con máxima de 21.4×, para la multiplicación de matrices, Jacobi y N-Reinas, respectivamente. Es claro que la mayor diferencia se da en N-Reinas, ya que justamente es la aplicación en la que el cluster de RPi obtiene el mejor rendimiento y la menor diferencia con el cluster de i5.

Teniendo en cuenta el objetivo planteado en esta tesina y los resultados obtenidos, se puede decir que los cluster de RPi ofrecen capacidades para cómputo paralelo y tasas de eficiencia energética superiores a los clusters de procesadores x86. Sin embargo, debido a la gran diferencia de rendimiento pico que tiene con estos últimos, no representan hoy una opción viable para HPC si el objetivo máximo es la reducción del tiempo de ejecución.

6.2. Trabajos futuros

Algunas de las líneas de trabajo futuro que se desprenden de esta tesina pueden ser:

- Recientemente, ha salido un nuevo modelo de RPi, el 3 B+. Como esta nueva placa incrementa la frecuencia del procesador a 1.4Ghz y la velocidad de la interfaz de red, resulta interesante replicar los experimentos realizados para poder cuantificar las mejoras introducidas.
- Considerando los resultados obtenidos y que existen otras placas similares pero más avanzadas que la RPi 3, interesa replicar el estudio realizado para continuar la búsqueda de alternativas viables para procesamiento paralelo basadas en procesadores ARM, como pueden ser la Banana Pi M3 [64], la Orange Pi Plus2 [65] o la ROCK64 [66].
- Como el consumo real puede diferir del consumo estimado, sería interesante medir realmente el consumo en ambos clusters para hacer un estudio más preciso del balance rendimiento/energía.

Referencias

[1] "TOP500." https://www.top500.org, 2017.

[2] M. Wolfe, "Arm yourselves for exascale - Part 1." https://www.hpcwire.com/2011/11/09/arm_yourselves_for_exascale_part_1, 2011.

[3] M. Jarus, S. Varrette, and A. O. . P. Bouvry, "Performance evaluation and energy efficiency of high-density hpc platforms based on intel, amd and arm processors," *Lecture Notes in Computer Science*, vol. 8046, pp. 182–200, 2013.

[4] "Sitio Web Beagleboard." https://beagleboard.org/, 2017.

[5] "Raspberry." https://www.raspberrypi.org, 2018.

[6] "PandaBoard." https://en.wikipedia.org/wiki/PandaBoard, 2017.

[7] N. Balakrishnan, *Building and benchmarking a low power ARM cluster*. PhD thesis, University of Edinburgh, 2012.

[8] H. Chaudhari, "Raspberry pi technology: A review," *International Journal of Innovative and Emerging Research in Engineering (IJIERE)*, pp. 83–87, 2015.

[9] B. Wilkinson and M. Allen, *Parallel Programming. Techniques and Applications Using Networked Workstations and Parallel Computers.* Prentice Hall, 2 ed., 2005.

[10] G. R. Andrews, *Foundations of Multithreaded, Parallel, and Distributed Programing.* Addison Wesley, 1 ed., 2000.

[11] A. Grama, G. Karypis, V. Kumar, and A. Gupta, *An Introduction to Parallel Computing. Design and Analysis of Algorithms.* Addison Wesley, 2 ed., 2003.

[12] L. Chai, Q. Gago, and D. K. Panda, "Understanding the impact of multi-core architecture in cluster computing: A case study with intel dual-core system," in

Proceedings of the IEEE International Symposium on Cluster Computing and the Grid, (RÃo de Janeiro (Brazil)), 2007.

[13] C. Zhang, X. Yuan, and A. Srinivasan, "Processor Affinity and MPI Performance on SMP-CMP Clusters," in *IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW)*, (Florida (USA)), 2010.

[14] G. Almasi and A. Gottlieb, *Highly Parallel Computing*. Benjamin Cummings, New York, 1994.

[15] I. Foster, "Compositional parallel programming languages," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 18, 1996.

[16] T. Rauber and G. Rünger., *Parallel Programming for Multicore and Cluster Systems*. Springer-Verlag Berlin Heidelberg, 2010.

[17] W. Stallings, *Computer organization and architecture (6. ed.)*. Prentice Hall, 2003.

[18] E. Rucci, *Evaluación de rendimiento y eficiencia energética de sistemas heterogéneos para bioinformática*. PhD thesis, Universidad Nacional de La Plata, 2017.

[19] T. R, V. M, and Z. P, *Parallel Computing. Numerics, Applications, and Trends*. Springer-Verlag London, 2009.

[20] O. S. Center, "MPI Primer / Developing with LAM," 1996.

[21] M. Naiouf, *Procesamiento Paralelo. Balance de Carga Dinámico en Algoritmos de Sorting*. PhD thesis, Facultad de Ciencias Exactas. Universidad Nacional de La Plata, 2004.

[22] B. Subramaniam and W. c. Feng, "The green index: A metric for evaluating system-wide energy efficiency in hpc systems," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops PhD Forum*, pp. 1007–1013, May 2012.

[23] P. B. Rao and S. K. Uma, "Raspberry pi home automation with wireless sensors using smart phone.," *International Journal of Computer Science and Mobile Computing*, vol. 4, pp. 797–803, 2015.

[24] V. Menezes, V. Patchava, and M. S. D. Gupta, "Surveillance and monitoring system using raspberry pi and simplecv," in *2015 International Conference on Green Computing and Internet of Things (ICGCIoT)*, pp. 1276–1278, Oct 2015.

[25] A. U. Bokade and V. R. Ratnaparkhe, "Video surveillance robot control using smartphone and raspberry pi," in *2016 International Conference on Communication and Signal Processing (ICCSP)*, pp. 2094–2097, April 2016.

[26] N. Gondchawar and R. S. Kawitkar, "Iot based smart agriculture.," *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 5, 2016.

[27] G. Jadhav, K. Jadhav, and K. Nadlamani, "Environment monitoring system using raspberry-pi," in *2016 International Research Journal of Engineering and Technology*, pp. 1168–1172, Abr 2016.

[28] A. S. Priya, S. Saranya, P. U. Maheshwari, N. S. Kumar, and N. G. Sherley, "Automatic detection and notification of potholes and humps on road and to measure pressure of the tire of the vehicle using raspberry pi," in *2016 International Journal of Scientific Research in Science, Engineering and Technology*, pp. 915–919, 2016.

[29] S. Mahajan, A. M. Adagale, and C. Sahare, "Intrusion detection system using raspberry pi honeypot in network security," in *2016 International Journal of Engineering Science and Computing*, pp. 2792–2795, Abr 2016.

[30] R. Haldar, S. Aruchamy, A. Chatterjee, and P. Bhattacharjee, "Diabetic retinopathy image enhancement using vessel extraction in retinal fundus images by programming in raspberry pi controller board," in *2016 International Conference on Inter Disciplinary Research in Engineering and Technology*, pp. 37–41, 2016.

[31] M. S. D. Gupta, V. Patchava, and V. Menezes, "Healthcare based on iot using raspberry pi," in *2015 International Conference on Green Computing and Internet of Things (ICGCIoT)*, pp. 796–799, Oct 2015.

[32] J. F. Kurose and K. W. Ross, *Redes de computadoras - Un enfoque descendente.* Pearson, 5 ed., 2010.

[33] M. P. Williams, "Solving polynomial equations using linear algebra," *Johns Hopkins APL Technical Digest*, vol. 28, no. 4, 2010.

[34] X. Wang and E. Serpedin, "An overview on the applications of matrix theory in wireless communications and signal processing," *Algorithms - Open Access Journal*, 2016.

[35] T. Akutsu, S. Miyano, and S. Kuhara, "Algorithms for identifying boolean networks and related biological networks based on matrix multiplication and fingerprint function," *Journal of computational biology*, vol. 7, pp. 331–343, 2000.

[36] I. Foster, "Designing and Building Parallel Programs," 1995.

[37] F. Tinetti, A. Quijano, and A. D. Giusti, "Heterogeneous networks of workstations and spmd scientific computing," in *1999 International Conference on Parallel Processing*, pp. 338–342, 1999.

[38] A. R. Tripathy and B. N. B. Ray, "Parallel algorithm for solving the system of simultaneous linear equations by jacobi method on extended fibonacci cubes," in *2013 3rd IEEE International Advance Computing Conference (IACC)*, pp. 970–976, Feb 2013.

[39] U. Kacar, M. Kirci, M. Kus, and E. O. Gunes, "An embedded biometric system," in *Proceedings of the 16th International Conference on Information Fusion*, pp. 736–742, July 2013.

[40] P. VanRaden, "Efficient methods to compute genomic predictions," *Journal of Dairy Science*, vol. 91, no. 11, pp. 4414 – 4423, 2008.

[41] Y. Lee, "Decision-aided jacobi iteration for signal detection in massive mimo systems," *Electronics Letters*, vol. 53, no. 23, pp. 1552–1554, 2017.

[42] A. Kumar, N. Agarwal, J. Bhadviya, and A. K. Tiwari, "An efficient 2-d jacobian iteration modeling for image interpolation," in *2012 19th IEEE International Conference on Electronics, Circuits, and Systems (ICECS 2012)*, pp. 977–980, Dec 2012.

[43] H. G. and W. G., *Introduction to high performance computing for scientists and engineers*. CRC Press is an imprint of Taylor 'I&' Francis Group, 2011.

[44] P. S. Segundo, "New decision rules for exact search in n-queens," *J. of Global Optimization*, vol. 51, pp. 497–514, Nov. 2011.

[45] A. Bruen and R. Dixon, "The n-queens problem," *Discrete Mathematics*, vol. 12(4), pp. 393–395, 1975.

[46] L. C. D. Giusti, P. Novarini, M. Naiouf, and A. E. D. Giusti, "Parallelization of the n-queens problem. load unbalance analysis.," *IX Congreso Argentino de Ciencias de la Computaciïżœn*, pp. 397–405, 2003.

[47] X. Zhang and Y. Yan, "Modeling and characterizing parallel computing performance on heterogeneous networks of workstations.," in *Seventh IEEE Symposium on Parallel and Distributed Processing*, pp. 25–34, 1995.

[48] Benchmark-tests, "Benchmark test intel core i5-4460." `http://benchmarks-tests.com/cpu_processors/amd_a10-6800k_vs_intel_core_i5-4460/power_consumption.php`.

[49] Hardware.Info, "Amd vs intel: 57 processor megatest." `https://goo.gl/hxZkTX`.

[50] F. D. Igual, C. García, G. Botella, L. Piñuel, M. Prieto-Matías, and F. Tirado, "Non-negative matrix factorization on low-power architectures and accelerators: A comparative study," *Computers 'I&' Electrical Engineering*, vol. 46, pp. 139–156, 2015.

[51] S. McDonald, "Raspberry pi 3 a first look." `http://blog.pimoroni.com/raspberry-pi-3`.

[52] R. P. Dramble, "Power consumption benchmarks." `https://www.pidramble.com/wiki/benchmarks/power-consumption`.

[53] T. M. Magazine, "Raspberry pi 3 is out now! specs, benchmarks & more." `https://www.raspberrypi.org/magpi/raspberry-pi-3-specs-benchmarks`.

[54] S. Desrochers, C. Paradis, and V. M. Weaver, "A validation of dram rapl power measurements," in *Proceedings of the Second International Symposium on Memory Systems*, MEMSYS '16, pp. 455–470, 2016.

[55] J. Kiepert, "Creating a raspberry pi-based beowulf cluster." `https://coen.boisestate.edu/ece/files/2013/05/Creating.a.Raspberry.Pi-Based.Beowulf.Cluster_v2.pdf`.

[56] B. Dye, *Distributed computing with the Raspberry Pi*. PhD thesis, Kansas State University, 2014.

[57] "Openfoam." `https://www.openfoam.com`.

[58] "Hpl." `http://www.netlib.org/benchmark/hpl`.

[59] S. Cox, J. Cox, R. Boardman, and et al, "Iridis-pi: a low-cost, compact demonstration cluster," *Cluster Computing*, vol. 17, p. 349358, June 2014.

[60] A. M. Pfalzgraf and J. A. Driscoll, "A low-cost computer cluster for high-performance computing education," in *IEEE International Conference on Electro/Information Technology*, pp. 362–366, June 2014.

[61] C. P. M. F. Cloutier and V. M. Weaver, "A raspberry pi cluster instrumented for fine-grained power measurement," *Electronics*, vol. 5, no. 4, 2016.

[62] V. A. Cicirello, "Design, configuration, implementation, and performance of a simple 32 core raspberry pi cluster," 08 2017.

[63] M. F. Cloutier, C. Paradis, and V. M. Weaver, "Design and analysis of a 32-bit embedded high-performance cluster optimized for energy and performance," in *2014 Hardware-Software Co-Design for High Performance Computing*, pp. 1–8, Nov 2014.

[64] "Banana Pi M3." `http://www.banana-pi.org/m3.html`, 2018.

[65] "Orange Pi Plus2." `http://www.orangepi.org/orangepiplus2/`, 2018.

[66] "ROCK64." `https://www.pine64.org/?page_id=7147`, 2018.

[67] I. Advanced Micro Devices, "Amd multic-core white paper," 2009.

[68] J. Al-Jaroodi, N. Mohamed, H. Jiang, and D. Swanson, "Modeling parallel applications performance on heterogeneous systems," in *Proceedings of the International Parallel and Distributed Processing Symposium 2003*, 2003.

[69] T. K. Atwood and D. J. Parry-Smith, *Introducción a la Bioinformática*. Prentice Hall, 2002.

[70] M. Ben-Ari, *Principles of Concurrent and Distributed Programming*. Addison Wesley, 2 ed., 2006.

[71] M. Bertogna, E. Grosclaude, M. Naiouf, A. D. Giusti, and E. Luque, "Dynamic on demand virtual clusters in grids," in *Euro-Par 2008 Workshops - Parallel Processing*, (Las Palmas de Gran Canaria (Spain)), 2008.

[72] A. Boukerche, A. C. M. A. de Melo, M. Ayala-Rincon, and T. M. Santana, "Parallel smith-waterman algorithm for local dna comparison in a cluster of workstations," in *Experimental and Efficient Algorithms. Proceedings of the 4th International Workshop WEA 2005.*, (Santorini Island, Greece), May 10-13 2005.

[73] T. Burger, "Intel Multi-Core Processors: Quick Reference Guide," 2005.

[74] C. Leopold, *Parallel and Distributed Computing. A survey of Models, Paradigms, and Approaches.* Wiley, 2001.

[75] Cloud Computing and Distributed Systems (CLOUDS) Laboratory. Department of Computer Science and Software Engineering. The University of Melbourne, Australia., "Cluster and grid computing," 12 2012.

[76] J. Dongarra, I. Foster, G. Fox, W. Gropp, K. Kennedy, L. Torczon, and A. White, *The Sourcebook of Parallel Computing.* Morgan Kauffman, 2003.

[77] N. Drosinos and N. Koziris, "Performance comparison of pure mpi vs hybrid mpi-openmp parallelization models on smp clusters," in *18th International Parallel and Distributed Processing Symposium (IPDPS'04) - Papers*, 2004.

[78] R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchinson, *Biological Sequence Analysis. Probabilistic models of proteins and nucleic acids.* Cambridge University Press, 7 ed., 2002.

[79] European Bioinformatics Institute, "What is bioinformatics?," 12 2012.

[80] R. Graham and G. Shipman, "MPI Support for Multi-core Architectures: Optimized Shared Memory Collectives," in *Proceedings of the 15th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, (Dublin (Ireland)), pp. 130 – 140, Springer-Verlag Berlin, Heidelberg, 2008.

[81] Hewlett-Packard Development Company, "HP BladeSystem."

[82] Hewlett-Packard Development Company , "HP BladeSystem c-Class architecture."

[83] Lawrence Livermore National Laboratory, "Mpi performance measurements," 12 2012.

[84] F. Leibovich, L. De Giusti, and M. Naiouf, "Parallel algorithms on clusters of multicores: Comparing message passing vs hybrid programming," in *Proceedings of the 2011 International Conference on Parallel and Distributed processing Techniques and Applications (PDPTA2011)*, 2011.

[85] W. S. Martins, J. B. D. Cuvillo, F. J. Useche, K. B. Theobald, and G. Gao, "A multithreaded parallel implementation of a dynamic programming algorithm for sequence comparison," in *Pacific Symposium on Biocomputing 2001*, 2001.

[86] M. McCool, "Scalable programming models for massively multicore processors," in *Proceeding of the IEEE*, vol. 96, pp. 816–831, 2007.

[87] R. C. F. Melo, M. E. T. Walter, A. C. M. A. Melo, R. Batista, M. Nardelli, T. Martins, and T. Fonseca, "Comparing two long biological sequences using a dsm system," in *Euro-Par 2003 Parallel Processing. Proceedings of the 9th International Euro-Par Conference.*, (Klagenfurt, Austria), August 26-29 2003.

[88] M. Miller, *Web-Based applications that change the way you work and collaborate online.* Que, 2009.

[89] U. o. I. a. U.-C. National Center for Supercomputing Applications, "Perfsuite default papi-derived metrics," 04 2013.

[90] M. Olszewski, J. Ansel, and S. Amarasinghe, "Kendo: Efficient determistic multithreading in software," in *ASPLOS '09 Proceedings of the 14th international conference on Architectural support for programming languages and operating systems.*, (New York (USA)), 2009.

[91] K. Olukotun, O. A. Olukotun, L. Hammond, and J. P. Laudon, *Chip Multiprocessors Architecture: Techniques to Improve Throughput and Latency.* Morgan & Claypool, 2007.

[92] R. Rabenseifner, "Hybrid parallel programming: Performance problems and chances," in *Proceedings of the 45th Cray User Group Conference*, 2003.

[93] E. Rucci, F. Chichizola, M. Naiouf, , and A. D. Giusti, "Performance comparison of parallel programming paradigms on a multicore cluster," in *Proceedings of*

the 24th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2012) (T. Gonzalez, ed.), vol. 1, (Las Vegas (USA)), pp. 216–221, Acta Press, 2012.

[94] E. Rucci, A. D. Giusti, F. Chichizola, M. Naiouf, and L. D. Giusti, "DNA sequence alignment: hybrid parallel programming on a multicore cluster," in *Recent Advances in Computers, Communications, Applied Social Science and Mathematics* (N. Mastorakis, V. Mladenov, B. Lepadatescu, H. R. Karimi, and C. G. Helmis, eds.), vol. 1, (Barcelona (Spain)), pp. 183–190, WSEAS Press, September 2011.

[95] S. Siddha, V. Pallipadi, and A. Mallick, "Process scheduling challenges in the era of multicore processors," *Intel Technology Journal*, vol. 11, no. 4, 2007.

[96] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *Journal of Molecular Biology*, vol. 147, pp. 195–197, 1981.

[97] M. D. Stefano, *Distributed data management for Grid Computing.* John Wiley & Sons Inc, 2005.

[98] S. I. Steinfadt, *Smith-Waterman sequence alignment for massively parallel high-performance computing architectures.* PhD thesis, Kent State University, 2010.

[99] X. Wu and V. Taylor, "Performance Characteristics of Hybrid MPI/OpenMP Implementations of NAS Parallel Benchmarks SP and BT on Large-scale Multicore Clusters," *The Computer Journal*, vol. 55, pp. 154–167, 2012.

[100] F. Zhang, X.-Z. Qiao, and Z.-Y. Liu, "A parallel smith-waterman algorithm based on divide and conquer," in *Proceedings of the Fifth International Conference on Algorithms and Architectures for Parallel Processing (ICA3PPï¿œ02)*, 2002.

[101] Z. Juhasz, P. Kacsuk, and D. Kranzlmuller, eds., *Distributed and Parallel Systems: Cluster and Grid Computing.* Springer Science + Business Media Inc., 2005.

[102] "ClustalW and Clustal X Multiple Sequence Alignment."

[103] "LAM/MPI." `http://www.lam-mpi.org/`, 2012.

[104] "The Message Passing Interface (MPI) standard." `http://www.mcs.anl.gov/research/projects/mpi`, 2012.

[105] "MPICH." http://www-unix.mcs.anl.gov/mpi/mpich2, 2012.

[106] "OpenMP.org." http://openmp.org/wp/, 2012.

[107] "OpenMPI." http://www.open-mpi.org/, 2012.

[108] "DHCP." https://tools.ietf.org/html/rfc2855, 2000.

[109] "NFS." https://tools.ietf.org/html/rfc1813, 1995.

[110] "The IP Network Address Translator (NAT)." https://tools.ietf.org/html/rfc1631, 1994.

[111] F. Tinetti and U. A. de Barcelona. Escuela Técnica Superior de Ingenieros. Departamento de Informática, *Cómputo paralelo en redes locales de computadoras.* Tesis inéditas, 2003.

[112] W. Feng, X. Feng, and R. Ge, "Green supercomputing comes of age," *IT Professional*, vol. 10, pp. 17–23, 2008.

[113] M. Crider, "APU, GPU, WTF? A guide to AMD s desktop processor line-up," 2015.

[114] K. Moammer, "Amd zen cpu microarchitecture details leaked in patch - doubles down on ipc and floating point throughput," 2015.

[115] K. Doucet and J. Zhang, "Learning cluster computing by creating a raspberry pi cluster," pp. 191–194, 04 2017.

[116] J. Saffran, G. García, M. Souza, P. H. Penna, M. Castro, L. Góes, and H. Freitas, "A low-cost energy-efficient raspberry pi cluster for data mining algorithms," 05 2017.

[117] A. Shinde, S. Pal, V. Singhvi, and M. Rana, "Cluster computing using raspberry pi," *IOSR Journal of Computer Engineering*, pp. 6–9, 2017.